Learning Term Rewriting Systems from Entailment

Hiroki Arimura[1,2], Hiroshi Sakamoto[1], Setsuo Arikawa[1]

[1] Dept. Informatics, Kyushu Univ., Fukuoka 812–8581, Japan
[2] Precursory Research for Embryonic Science and Technology at PRESTO, Japan Science and Technology Corporation
{arim,hiroshi}@i.kyushu-u.ac.jp

Abstract. Learning from entailment is a learning model suitable for learning first-order formulas, in which examples are formulas in the first-order language which are implied or not implied. In this paper, we study the learnability of term rewriting systems in this framework. A term rewriting system is a simple model of tree transduction used for semi-structured data and Web, which can be considered as a fragment of first-order equational logic with a binary predicate $\Rightarrow$. First, we present a generic algorithm using entailment equivalence queries and request for hint queries, and show that the class of terminating term rewriting systems is exactly learnable by this algorithm. Next, we present a polynomial time learning algorithm using entailment equivalence and entailment membership queries for another subclass k-variable linear tree translations with top-down rewriting strategy.

1 Introduction

In this paper, we study the learnability of functions and relations represented by *term rewriting systems* in the framework of *learing from entailment*. A term rewriting system (TRS, or rewriting systems) is a set of ordered equations $(s \Rightarrow t)$ such as

$$\left\{ \begin{array}{lll} top(push(x,y)) & \Rightarrow & x, \\ pop(push(x,y)) & \Rightarrow & y, \\ ser(tree(x,y)) & \Rightarrow & app(ser(x), ser(y)) \end{array} \right\},$$

and defies a function over first-order terms using the normal form of rewriting. For example, the term $top(pop(push(a, push(b, \Lambda))))$ reduces to b. By the following axioms, rewriting systems can be regarded as a fragment of ordered equational logic with the only binary predicate $\cdot \Rightarrow \cdot$:

- Reflexivity: $(\forall x)\ (x \Rightarrow x)$.
- Transitivity: $(\forall x, y, z)\ (x \Rightarrow y)\ \&\ (y \Rightarrow z) \rightarrow (x \Rightarrow z)$.
- Congruence: $(\forall \mathbf{x}, \mathbf{y})\ \wedge_{1 \le i \le k} (x_i \Rightarrow y_i) \rightarrow (f(\mathbf{x}) \Rightarrow f(\mathbf{y}))$ for every $k \ge 1$ and function symbol f of arity k, where $\mathbf{x} = x_1, \ldots, x_k$ and $\mathbf{y} = y_1, \ldots, y_k$.

It is well known that if a term rewriting system is terminating and confluent then the rewriting relation becomes a total function. The function defined by a term rewriting system is a simple model of functional programming languages, and furthermore, closely related to the recent development of tree rewriting technique used for semi-structured data such as XML Translations or XSLT language[6].

The learning model considered in this paper is learning from entailment [1, 9, 12]. Learning from entailment is a variant of query learning model [2] tailored for learning first-order formulas, in which examples are formulas in the first-order language which are implied or not implied. In this framework, a number of efficient learnability results are obtained for various fragments of first-order logic have been shown to be learnable using this framework [1, 4, 9–12, 15, 16].

Recently, it is found that a class of learning algorithms for subclasses of first-order Horn programs has a common scheme [16, 22], which has its origin in a monotone Boolean DNF learner in [18]. Reddy and Tadepalli [16] proposed an algorithm with subsumption membership queries and saturation generalization. Yamamoto and Arimura [22] gave a generic bottom-up learning algorithm (Fig. 1 in this paper) that learns a subclass of Horn programs using subsumption queries and entailment equivalent queries under a certain condition based on the works of Yamamoto on proof completion [19, 21].

They have demonstrate that known efficient learnability results for several fragments of first-order Horn sentences can be derived from this generic algorithm by appropriately selecting specific procedures for inverting resolution: unit clause programs[4], non-recursive Horn definitions [15], constrained atoms with acyclic background theory[12], propositional Horn sentences[9], constrained acyclic Horn programs[4], and acyclic Horn programs [16].

In this paper, we apply this framework based on the proof completion to polynomial time learning of functions over labeled trees expressed by term rewriting systems. First in Section 3, we consider the learning problem for the general class of terminating term rewriting systems. As an instance of the generic algorithm mentioned above, we give a learning algorithm that learns terminating term rewriting systems using $O(mn)$ entailment equivalence queries and $O(mn\max(n,l))$ request for hint queries, where $m = |H_*|$, n is the size of the largest counterexample, and l is the length of the longest rewriting sequence for counterexamples returned by queries. Although the algorithm is not efficient in general, we show that if any rewriting sequence in the rewriting relation is polynomially bounded for the target class, then the algorithm learns the target hypothesis in polynomial time.

Next, in Section 4, we consider the polynomial time learnability for a small subclass, called *k-variable linear tree translations*, which capture a top-down computation by simple rules with restricted syntax such as in Web and XML applications. We show that there exists an algorithm that exactly learns any target hypothesis H_* in this class in polynomial time using $O(m)$ entailment equivalence queries and $O(mn^{2k})$ entailment membership queries, where $m = |H_*|$ and n is the size of the largest counterexample.

In Section 5, we summarize the results, and state some future research problem in application of this technique to information extraction from Web and semistructured data. A part of Section 4 will be presented in [17].

2 Preliminaries

In this section, we introduce basic definitions and notations on the first-order equational logic and term rewriting systems according to [7].

2.1 First-order terms

For a set A, we denote by A^* the set of all finite sequences over A and by $|A|$ the cardinality of A. The empty sequence ε is the sequence of length zero. Let $\mathbf{N} = \{0, 1, 2, \ldots\}$ denotes the set of nonnegative integers.

Let $\Sigma = \cup_{n \geq 0} \Sigma_n$ be a finite *ranked alphabet* of *function symbols*, called a *signature*, where each $f \in \Sigma_n$ is associated with arity n. Let X be a countable set of *variables* disjoint with Σ. A function symbol of arity zero is called a *constant*. We assume that Σ contains at least one constant.

We denote by $\mathcal{T}(\Sigma, X)$ and $\mathcal{G}(\Sigma)$ respectively the sets of all *first-order terms* (or *terms*, for short) and the set of all *ground* terms over Σ. If alphabets Σ and X are understood from context, then we may write $\mathcal{T}$ and $\mathcal{G}$ instead of $\mathcal{T}(\Sigma, X)$ and $\mathcal{G}(\Sigma)$.

Example 1. Let $\Sigma_0 = \{0, null\}$, $\Sigma_1 = \{top, pop, succ\}$, $\Sigma_2 = \{push, alternate\}$, and $X = \{x, y, z\}$. Then, expressions $x, 0, null, push(0, null), top(push(x, y))$, $push(succ(x), push(x, null))$ are terms of $\mathcal{T}(\Sigma, X)$. We also see that 0 and $push(0, null)$ are members of $\mathcal{G}(\Sigma)$.

We can identify a term $t \in \mathcal{T}$ with a labeled ordered tree as follows. A leaf is labeled with either a variable $x \in X$ or a constant $c \in \Sigma_0$. An internal node is labeled by a function symbol $f \in \Sigma$ of positive arity which equals the out-degree of the node. The children of the node are ordered from left to right. Each each node has a *position*, which is a sequence of nonnegative integers in $\mathbf{N}^*$ defined inductively as follows. The root of T has position ε. If an internal node of out-degree $k > 0$ has position $\pi \in \mathbf{N}$ then its i-th child has the position $\pi.i$ for each $1 \leq i \leq k$. We denote by $Occ(t)$ the set of all positions in t. We may call π the node of t.

Let s, t be terms. For position $\pi \in Occ(t)$, we denote by t/π the subterm $s \in \mathcal{T}$ of t at π, i.e., s is the subterm of t whose root is π. If $s = t/\pi$ then we say that either s *occurs at* π in t, or π is the *occurrence* of s in t. We denote by $t[s]_\pi$ the *term obtained from term t by replacing the subterm t/π with term s*. A *context*, denoted by $C[\cdot]$, is a term in $\mathcal{T}(\Sigma \cup \{\$\}, X)$ that contains exactly one occurrence of a special marker $\$ \notin \Sigma$ of arity zero. Let $C[\cdot]$ be a context with \$ at a position π. Then for term t, we define $C[t] = C[t]_\pi$.

Example 2. For example, if $t = push(0, pop(push(y, z)))$ then $t/2.1$ is the first argument of the second argument of t, that is, $push(y, z)$. If $s = null$ then we see $t[s]_{2.1} = push(0, pop(null))$. For a context $C[\cdot] = push(succ(0), \$)$, $C[push(0, null)] = push(succ(0), push(0, null))$.

A *substitution* θ is a homomorphism from $\mathcal{T}$ to themselves, and the application of θ to term t is denoted by $t\theta$.For an ordered equation $C = (s \Rightarrow t)$, we define $C\sigma = (s\sigma \Rightarrow t\sigma)$.

The *subsumption relation* is a binary relation $\succeq$ defined by $s \succeq t$ iff $s\theta = t$ for some substitution θ. If $s \succeq t$ then we say that s *subsumes* t, or t is an *instance of* s. Let C, D be ordered equations. If $C\sigma = D$ for some substitution σ then we define $C \succeq D$ and say C *subsumes* D. For rewriting system H, if there exists some $C \in H$ such that $C \succeq D$ then we write $H \succeq D$.

2.2 Rewriting systems

In this paper, the basic object in our logic is an ordered pair $(s, t) \in \mathcal{T} \times \mathcal{T}$ of terms. In what follows, we write $(s \Rightarrow t)$ for the ordered pair and call it an *ordered equation* or *rewriting*. An ordered equation $s \Rightarrow t$ is called *strict* if $var(l) \supseteq var(r)$ holds.

Definition 1 A *rewriting system* is a finite set $E = \{\, l_i \Rightarrow r_i \mid 1 \leq i \leq m \,\}$ of strict ordered equations in $\mathcal{T} \times \mathcal{T}$, where $m \geq 0$. Elements $l_i \Rightarrow r_i \in E$ are called *rewriting rule* (or *rule*, for short).

Example 3. Let Σ be the signature of Example. 2. The following is a rewriting system that represents the data structure for stacks.

$$
\begin{array}{lcl}
top(push(x,y)) & \Rightarrow & x \\
pop(push(x,y)) & \Rightarrow & y \\
alternate(\Lambda, z) & \Rightarrow & z \\
alternate(push(x,y),z) & \Rightarrow & push(x, alternate(z,y))
\end{array}
$$

Next, we introduce the semantics $\mathbf{M}(H)$, the rewriting relation defined by a rewriting system H.

Definition 2 Let E be a rewriting system. The *rewriting relation* induced in E is a binary relation $\underset{E}{\rightarrow}$ defined as follows: for any terms $s, t \in \mathcal{T}$, $s \underset{E}{\rightarrow} t$ iff there exist some rule $(l \Rightarrow r) \in E$, some position $\pi \in Occ(s)$, and some substitution σ such that t is obtained from s by replacing the subterm s/π that matches the left hand side $l\sigma$ with the right hand side $r\sigma$, i.e., $s/\pi = l\sigma$ and $t = s[r\sigma]_\pi$.

In the definition above, the subterm s/π to be replaced is called a *redex* in s. If $s \underset{E}{\rightarrow} t$ then s is said to *reduce to* t by E. We often write $t \underset{E}{\leftarrow} s$ for $s \underset{E}{\rightarrow} t$. The first semantics is the relation $\mathbf{M}(E)$ that realized by a rewriting system E. For binary relations R, S, we denote by R^* the transitive and reflexive closure of R and by $R \circ S = \{\, (a, c) \mid (a, b) \in R, (b, c) \in S \,\}$ the composition of R and S.

Definition 3 Let E be a rewriting system and $s, t \in \mathcal{T}$ be terms. By definition, if $s \overset{*}{\underset{E}{\rightarrow}} t$ then there exists a finite sequence $t_0 = s \underset{E}{\rightarrow} t_1 \underset{E}{\rightarrow} \cdots \underset{E}{\rightarrow} t_n = t$ of terms in $\mathcal{T}$ starting and ending with s and t, respectively. This sequence is called a *derivation for* $s \Rightarrow t$ by E. We denote the relation $\overset{*}{\underset{E}{\rightarrow}}$ by $\mathbf{M}(E)$.

In other words, $s \xrightarrow[E]{*} t$ iff t is obtained from s by successive applications of rules in E. The length of this derivation is defined to be n.

Example 4. Let E be the rewriting system of Example. 3. The following is an example of derivation by E, where underlines indicate the redexes.

$$\begin{array}{rcl}
alternate(push(\underline{top(push(0,\Lambda))},\Lambda),\Lambda) & \xrightarrow[E]{} & \underline{alternate(push(0,\Lambda),\Lambda)} \\
& \xrightarrow[E]{} & push(0,\underline{alternate(\Lambda,\Lambda)}) \\
& \xrightarrow[E]{} & push(0,\Lambda)
\end{array}$$

Example 5. Let us show another example. The following is a rewriting system that represents the serialization of a binary tree.

$$\begin{array}{lcl}
ser(c) & \Rightarrow & push(c,\Lambda) \text{ for each } c \in \Sigma \\
ser(tree(x,y)) & \Rightarrow & app(ser(x),ser(y)) \\
app(\Lambda,z) & \Rightarrow & z \\
app(push(x,y),z) & \Rightarrow & push(x,app(y,z))
\end{array}$$

Then, we show a derivation for $ser(tree(tree(a,b),c))$ by E below.

$$\begin{array}{rcl}
\underline{serial(tree(tree(a,b),c))} & \xrightarrow[E]{} & app(\underline{serial(tree(a,b))},serial(c)) \\
& \xrightarrow[E]{} & app(app(serial(a),\underline{serial(b)}),serial(c)) \\
& \xrightarrow[E]{*} & app(\underline{app(push(a,\Lambda),push(b,\Lambda))},serial(c)) \\
& \xrightarrow[E]{} & app(push(a,\underline{app(\Lambda,push(b,\Lambda))}),serial(c)) \\
& \xrightarrow[E]{} & \underline{app(push(a,push(b,\Lambda),serial(c))} \\
& \xrightarrow[E]{} & push(a,\underline{app(push(b,\Lambda),serial(c))}) \\
& \xrightarrow[E]{} & push(a,push(b,\underline{app(\Lambda,serial(c))})) \\
& \xrightarrow[E]{} & push(a,push(b,\underline{serial(c)})) \\
& \xrightarrow[E]{} & push(a,push(b,push(c,\Lambda)))
\end{array}$$

We give a simple characterization of our relational semantics $\mathbf{M}(E)$, which is useful for designing a learning algorithm. If $C = (s \Rightarrow t)$ and $D = (u[s] \Rightarrow u[t])$ for some context $u[\cdot]$ then we define $C \trianglerighteq D$ and say C *embeds in* D.

Lemma 1. *For any rewriting system $H*$, $\mathbf{M}(H*)$ $(= \xrightarrow[H_*]{*})$ is the smallest binary relation containing H_* that is closed under subsumption, embedding, and composition, i.e., $\mathbf{M}(H_*)$ $(= \xrightarrow[H_*]{*})$ satisfies the following three conditions:*

- *Subsumption: If $C \in \mathbf{M}(H_*)$ and $C \succeq D$ then $D \in \mathbf{M}(H_*)$.*
- *Embedding: If $C \in \mathbf{M}(H_*)$ and $C \trianglerighteq D$ then $D \in \mathbf{M}(H_*)$.*
- *Composition: If $(s \Rightarrow y), (u \Rightarrow t) \in \mathbf{M}(H_*)$ then $(s \Rightarrow t) \in \mathbf{M}(H_*)$.*

Definition 4 A rewriting system E is said to be *terminating* if for every term $s \in \mathcal{T}$, there is no infinite sequence $t_0 = s \xrightarrow[E]{} t_1 \xrightarrow[E]{} \cdots \xrightarrow[E]{} t_i \xrightarrow[E]{} \cdots$, $i \geq 0$, starting from s.

We denote by TRS_T the classes of terminating TRS.

2.3 The Learning Model

Our problem is learning an unknown rewriting system H_* from examples of ordered pairs $s \Rightarrow t$ that $\mathbf{M}(H_*)$ entails or does not entail. As a formal model, we employ *learning from entailment* [1, 9] tailored for rewriting systems, which is a variant of exact learning model [2].

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

A *learning algorithm* $\mathcal{A}$ may collect the information about H_* using the following type of queries.

Definition 5 An *entailment equivalence query* (EQ) is to propose any rewriting 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 rewriting $C \in U$ as a *counterexample* such that either $C \in \mathbf{M}(H_*)\backslash\mathbf{M}(H)$, or $C \in \mathbf{M}(H)\backslash\mathbf{M}(H_*)$. A counterexample is *positive* if $C \in \mathbf{M}(H_*)$ and *negative* if $C \notin \mathbf{M}(H_*)$.

Definition 6 An *entailment membership query* (MQ) is to propose a clause $C \in U$. The answer to the membership query is "yes" if $C \in \mathbf{M}(H_*)$, and "no" otherwise.

As a tool for designing a learning algorithm for TRS, we use augmented versions of membership queries, called subsumption membership queries and request for hint queries, which allows us to collect detailed information on the structure of derivations by TRS. Request for hint queries are originally introduced by Angluin [1] for learning of propositional Horn clauses.

Definition 7 A *subsumption membership query* (SQ) is to propose a clause $C \in U$, whose answer is "yes" if $H_* \succeq C$, and "no" otherwise.

Definition 8 (Angluin [1]) A *request for hint query* (RQ) is an augmented MQ query which tells whether a given ordered equation C is subsumed, entailed (but not subsumed), or not entailed by H_*. If $C \in \mathbf{M}(H_*)$ but $C \not\preceq H_*$ then the query receives as a *hint* a term u that appears in a derivation for $C = (s \Rightarrow t)$ but is distinct from both s and t.

Now we state our learning problem.

Definition 9 (Angluin [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 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 we can transform any exact learning algorithm with equivalence queries into a *polynomial time PAC-learnability* or a *polynomial time online learnability* by a standard transformation under a mild condition on $\mathcal{H}$. Further, this is true whether membership queries are allowed or not [2].

3 Learning algorithm with request for hint queries

In this section, we present a generic algorithm that exactly learns any rewriting systems H_* using entailment equivalence and a kind of entailment membership queries, called request for hint queries, under the relational semantics $\mathbf{M}(H_*)$.

3.1 Basic strategy

Our strategy to learn an unknown program H_* is explained as follows. Our learning algorithm *EntLearn* is an algorithm that, starting with an initial hypothesis H_0, iterates the process of making $EQ(H_n)$, receiving a counterexample E_n, and then updating H_n using E_n and MQ to produce the next hypothesis H_{n+1}.

In our strategy, we set H_0 to be the most specific hypothesis $\emptyset$ and *EntLearn* try to produce a sequence $H_0, H_1, \ldots, H_n, \ldots$ of hypotheses such that $\mathbf{M}(H_{n-1}) \subset \mathbf{M}(H_n)$ and $\mathbf{M}(H_n) \subseteq \mathbf{M}(H_*)$ for any $n \geq 0$. By the monotonicity, this sequence will eventually converges to H_*.

To ensure the convergence, we use an approximation $\cdot \sqsubseteq \cdot$ of $\mathbf{M}(\cdot) \subseteq \mathbf{M}(\cdot)$ defined as follows. We extend the subsumption relation $\succeq$ for ordered equations to the relation $\sqsupseteq$ for sets of ordered equations.

Definition 10 (Arimura et al.[3]) Let P, Q be sets of rewriting rules. Then, we define $P \sqsubseteq Q$ iff for any $C \in P$, there exists some $D \in Q$ such that $C \preceq D$. We say that Q is a *generalization* of P, or P is a *refinement* of Q. If $P \sqsubseteq Q$ but $Q \not\sqsubseteq P$ then we define $P \sqsubset Q$.

Lemma 2. *$P \sqsubseteq Q$ implies $\mathbf{M}(P) \subseteq \mathbf{M}(Q)$.*

Proof. Since $\mathbf{M}(Q)$ is closed under subsumption, if $P \sqsubseteq Q$ then $P \subseteq \mathbf{M}(Q)$. By Lemma 1, $\mathbf{M}(Q)$ is closed under subsumption, embedding, composition. On the other hand, $\mathbf{M}(P)$ is the smallest set containing P that is also closed under the three operations above. Hence, we conclude that $\mathbf{M}(P) \subseteq \mathbf{M}(Q)$. □

Note that the converse of Lemma 2 does not hold in general. By Lemma 2, $\cdot \sqsubseteq \cdot$ is a coarse approximation of $\mathbf{M}(\cdot) \subseteq \mathbf{M}(\cdot)$.

3.2 The learning algorithm

In Fig. 1, we give our basic learning algorithm *EntLearn* for learning H_* using EQ and SQ, where procedures $Missing(E, H)$ and $Gen(C, D)$ will be specified below. The subsumption membership queries will be replaced with request for hint queries.

```
Algorithm EntLearn;
  H := ∅;
  while EQ(H) returns "no" do begin    /* H_* ≢ H */
    Let E be a counterexample returned by the query;
    D := Missing(E, H);
    foreach C ∈ H do make query SQ(Gen(C, D));
    if the query returns "yes" for some C ∈ H then
      H := (H − {C}) ∪ {Gen(C, D)});
    else
      H := H ∪ {D};
  end /* while */
  return H;
```

Fig. 1. A learning algorithm for terminating rewriting systems using entailment equivalence queries and subsumption queries.

Definition 11 Let H be rewriting systems. A rule C is said to be *missing* with respect to H if C is an instance of some clause in H_* and C is not entailed by H, i.e., $C \preceq H_*$ and $C \notin \mathsf{M}(H)$.

It is easy to see that if $H \sqsubset H_*$ then there always exists some ordered equation D that is missing in H_* w.r.t. H.

Definition 12 Procedures $Missing(E, H)$ and $Gen(C, D)$ in Fig. 1 are any procedures satisfying the following conditions:

(a) $Missing(E, H)$ denotes a procedure that returns a missing clause in H_* w.r.t. H. By definition, the missing clause is also an instance of some missing clause in H_*.
(b) $Gen(C, D)$ is a procedure that returns an instance B of some rule in H_* such that B subsumes both C and D if it exists. If no such B exists then $Gen(C, D)$ can return any ordered equation.

The algorithm *EntLearn* in Fig 1 is a descendant of a PAC-learning algorithm for monotone DNF formulas in Valiant [18] tailored for term rewriting systems. The implementation of the procedures not yet specified above will be given in the rest of this section.

3.3 Inverting rewriting steps

In this section, we implement two procedures $Missing(E, H)$ and $Gen(C, D)$ which are not specified in *EntLearn* of Fig. 1 by introducing a set of operators that inverts rewriting steps.

Recall that $\mathsf{M}(H_*)$ is the smallest relation containing H_* that is closed under three operations: subsumption, embedding, and composition (Lemma 1).

Procedure Missing. First, we describe the implementation of the procedure *Missing*. Clearly, any missing ordered equation D is a positive counter example, but not necessarily all positive counterexample are missing in H_*. Thus, the first

```
Procedure Missing(E, H);
  while true do
    if Shrink(E, H) returns any result D then E := D;
    else if Strip(E, H) returns any result D then E := D;
    else break the while loop;
  return E;

Procedure Shrink((s ⇒ t), H);
  if RQ(s ⇒ t) returns a hint u then
    if s ⇒ u ∉ M(H) then t := u;
    else if u ⇒ t ∉ M(H) then s := u;
    return s ⇒ t;
  else
    return "failed";

Procedure Strip((s ⇒ t), H);
  if (∃i ∈ Occ(s) ∩ Occ(t))  [ (s/i ⇒ t/i) ∉ M(H) and MQ(s/i ⇒ t/i) = "yes" ] then
    return s/i ⇒ t/i;
  else
    return "failed";
```

Fig. 2. Procedures for finding a missing rule instance using request for hint query

task of $EntLearn$ is to reconstruct some missing ordered equation D from a given counter example E_n.

In Fig. 2, we show the operators $Shrink$ and the $Strip$ for implementing the procedure $Missing$, which invert the composition and the embedding operations, respectively. In the procedure $Shrink$, we use the following request for hint queries (RQ).

Lemma 3. *Let H_* be the terminating rewriting system, $H \sqsubseteq H_*$ the current hypothesis and E a positive counterexample $E \in \mathbf{M}(H_*) \backslash \mathbf{M}(H)$. The procedure $Missing(E, H)$ of Fig. 2 returns a missing ordered equation D of H_* w.r.t. H in the polynomial time in $O(\max(n, p(E)))$, where $p(E)$ is the length of the longest derivation for E by H_* and n is the maximum depth of E.*

Proof. Let E_0 be the initial input. Whenever $Missing$ enters the while loop, either the length of the longest derivation for E or the maximum height of E properly decreases. Thus, $Missing$ terminates after $O(\max(n, p(E)))$ executions of the while loop. Let D be the answer returned by $Missing(E, H)$. By construction of the algorithms, we can easily see that $D \in \mathbf{M}(H_*) \backslash \mathbf{M}(H)$ still holds. Suppose contrary that D is not subsumed by H_*. Since $D \in \mathbf{M}(H_*)$, either every derivation for D is length more than one or $C \trianglerighteq D$ for some missing $C \in H_*$ w.r.t. H, where C is embedded at the strict inner position of D. In the former case, it must be that RQ must return a hint, and thus, $Shrink$ will continue the while loop. In the latter case, $Strip$ may succeed. These contradict that $Missing$ has terminated. Hence, the result follows. □

Procedure Gen. Once we found a missing clause D of H_*, we update H_n by merging D with some clause $C \in H_n$. Otherwise, we simply add D into H_n. At this step, we must be careful not to over-generalize C and thus H_n because once over-generalization happens, the condition $\mathbf{M}(H_n) \subseteq \mathbf{M}(H_*)$ will be violated. To do this, we use the least generalization operator, which is originally introduced by Plotkin [13].

Definition 13 (Plotkin [13]) For a pair D_1, D_2 of ordered equations, a *common generalization* of D_1, D_2 is an ordered equation C such that $D_i \preceq C$ for every $i = 1, 2$. A *least general generalization* of P, denoted by $lg(D_1, D_2)$, is a common generalization C of D_1, D_2 such that $C \preceq D$ for every common generalization D of D_1, D_2.

Lemma 4. (Plotkin [13]) *For a pair D_1, D_2 of ordered equations, the least generalization $lg(D_1, D_2)$ is unique modulo renaming, of linear size, and polynomial time computable in the total size of D_1 and D_2.*

Lemma 5. *Let the procedure $Gen(C, D)$ returns $lg(C, D)$ for input ordered equations C, D. Then, $Gen(C, D)$ correctly satisfies the conditions in (b) of Definition 12.*

Proof. Obvious from the definition of $lg(C, D)$. □

Queries SQ and MQ. We can implement subsumption membership queries SQ in EntLearn (Fig. 1) and entailment membership queries in Shrink (Fig. 2) using request for hint queries as follows. The correctness is obvious from the definition.

```
Procedure SQ(C);
  if (RQ(C) returns "subsumed") then
    return "yes";
  else return "no";

Procedure MQ(C);
  if (RQ(C) returns either "entailed" or "subsumed") then
    return "yes";
  else return "no";
```

Fig. 3. Procedures for replacing a subsumption membership query and an entailment membership query with a request for hint query

3.4 Correctness and time complexity

Now we show the correctness of our learning algorithm *EntLearn*. Recall that starting with an initial hypothesis H_0, *EntLearn* iterates the process of making $EQ(H_n)$, receiving a counterexample E_n, and then updating H_n using E_n and MQ to produce the next hypothesis H_{n+1} for stages $n = 0, 1, 2, \ldots$.

Definition 14 (Arimura et al.[5]) A refinement P of Q is said to be *conservative* if for any $D \in Q$ there is at most one $C \in P$ such that $C \preceq D$.

Lemma 6. *Let $E_0, E_1, \ldots, E_n, \ldots$ and $H_0, H_1, \ldots, H_n, \ldots$ be a sequence of counterexamples received and a sequence of hypotheses that produced by EntLearn, respectively. Then for every $i \geq 0$, the following (i) – (iii) holds:*

(i) *E_i is a positive counterexample.*
(ii) *H_i is a conservative refinement of H_*.*
(iii) *$H_0 \sqsubset H_1 \sqsubset \cdots \sqsubset H_i \sqsubset \cdots$.*

Lemma 7. *Let Δ be an increasing sequence $H_0 \sqsubset H_1 \sqsubset \cdots \sqsubset H_i \sqsubset \cdots$ of rewriting systems such that H_i is a conservative refinement of H_*. Then, the length of Δ is finite, and furthermore bounded by mn, where $m = |H_*|$ and n is the size of the largest ordered equation in Δ.*

Theorem 1. *The algorithm EntLearn exactly identifies every terminating rewriting system $H_* \in TRS_T$ consisting of m rules in time polynomial time in m and n making $O(mn)$ entailment equivalence queries and $O(mn \max(n, l))$ request for hint queries, where n is the size of the longest counterexample and l is the length of the longest derivation for a counterexample seen so far.*

Note that the maximum length l of derivations is finite but may not be bounded by any polynomial in general for terminating rewriting systems. For rewriting systems whose polynomial time termination is ensured by polynomial method [7], the learning algorithm will work in polynomial time in m and n above.

4 Polynomial time learning for a subclass of rewriting systems

In this section, we introduce a subclass called k-variable linear tree translations by restricting simultaneously the occurrences of variables appearing in a rule and the rewriting strategy. Then, we show that the class is polynomial time learnable using entailment equivalence and entailment membership queries. Let $p[x_1, \ldots, x_l]$ denotes a term $p \in \mathcal{T}$ that contains variable x_i exactly once in p for every $1 \leq i \leq l$.

Definition 15 Let $k \geq 0$ be any nonnegative integer. A *k-variable linear rule* is a rewriting rule of the form

$$C = (p[x_1, \ldots, x_l], q[x_1, \ldots, x_l]),$$

where both the left hand side p and the right hand side q contain exactly $l \geq k$ variables $x_1, \ldots, x_l$ ($l \leq k$). Each variable x_i must occur exactly once in each side of C. A *k-variable linear translation system* is a set of k-variable linear rules.

Definition 16 For every $k \geq 0$, we denote by $LTT(k)$ the class of all k-variable linear tree translation systems and define $LTT = \cup_{k \geq 0} LTT(k)$.

```
Algorithm Learn_LTT;
Input: positive integer k;
Given: the equivalence and the membership queries for the target set H_* ∈ LTT(k);
Output: a set H of linear tree translations equivalent to H_*;
begin
  H := ∅;
  until EQ(H) returns "yes" do
  begin
    Let E be a counterexample returned by the equivalence query;
    D := StripAll(E, H);          /* Finding a missing clause */
    H := H ∪ Expand(D, k);        /* Adding all possible candidates correct in T(H_*) */
  end /* main loop */
  return H;

Procedure StripAll((s⇒t), H);
  Output: a smallest positive counterexample obtained from (s⇒t) with respect to ⊵;
  while (∃i ∈ Occ(s) ∩ Occ(t)) [ (s/i⇒t/i) ∉ M(H) and MQ(s/i⇒t/i) = "yes" ] do
    (s, t) := (s/i⇒t/i);
  return (s,t);

Procedure Expand(D, k);
  Output: the set of all correct k-linear rules with respect to T(H_*) that covers D;
  return { C ∈ M(H_*) | C ⪰_{H_*} D, C is k-variable linear };
```

Fig. 4. A learning algorithm for k-variable linear tree translations using equivalence and membership queries, and its subprocedures

Definition 17 Let $H \in LLT(k)$ be a k-variable linear rewriting system. We define the binary relation $\mathbf{T}(H) \subseteq \mathcal{T} \times \mathcal{T}$ 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 $(p_i, q_i) \in \mathbf{T}(H)$ for every i then $(f(p_1, \ldots, p_k), f(q_1, \ldots, q_k)) \in \mathbf{T}(H)$.
- *Parallel Application*: If $(l[x_1, \ldots, x_l], r[x_1, \ldots, x_l]) \in H$ is a k-variable linear rule in H and $(p_i, q_i) \in \mathbf{T}(H)$ for every i then $(l[p_1, \ldots, p_k], r[q_1, \ldots, q_k]) \in \mathbf{T}(H)$.

The strategy introduced here allows only the top-down rewriting with parallel applications. More precisely, this semantics with $\mathbf{T}(H)$ corresponds to the computation of top-down tree transducer [8].

Lemma 8. *For every $H \in LTT(k)$, $\mathbf{T}(H) \subseteq \mathbf{M}(H)$.*

The *reachability problem* or *membership problem* on a binary relation $R(H)$ is, given rewriting system H and ordered equation C, to decide if $C \in R(H)$ holds [7]. The following lemma says that the reachability problem on $\mathbf{T}(E)$ for $LTT(k)$ is efficiently decidable, while that on $\mathbf{M}(E)$ is undecidable in general for term rewriting systems.

Lemma 9. *The membership problem on $\mathbf{T}(H)$ is polynomial time decidable.*

Proof. We can decide if $C \in \mathbf{T}(H)$ in time $O(mn^5)$, where $m = |H|$ and n is the size of C, by computing all pairs of positions (π, τ) in $C = (p, q)$ such that $(p/\pi, q/\tau) \in \mathbf{T}(H)$ using dynamic programming. Details are omitted. □

In Fig. 4, we give a polynomial time learning algorithm *Learn_LTT* for learning k-variable linear tree translations using entailment equivalence and entailment membership queries only. For details of the procedure and the proofs of its correctness, please consult [17].

Definition 18 Let $C = (p[x_1, \ldots, x_l], q[x_1, \ldots, x_l])$, $l \geq k$, be k-variable linear rule, D be any rule, and H_* be an unknown rewriting rule. C is said to *cover* D *relative to* H_*, denoted by $C \succeq_{H_*} D$, if there exist $(p_i, q_i) \in \mathbf{M}(H_*)$, $i = 1, \ldots, l$, such that $D = (p[p_1, \ldots, p_l], q[q_1, \ldots, q_l])$.

Theorem 2 ([17]). *For every $k \geq 0$, the algorithm Learn_LTT of Fig. 4 exactly identifies any k-variable linear tree translation $H_* \in LTT(k)$ consisting of m rules in time polynomial time in m and n making $O(m)$ entailment equivalence queries and $O(kn^{2k})$ entailment membership queries, where n is the size of the longest counterexample.*

5 Conclusion

In this paper, we have studied the learnability of term rewriting systems in the framework of learning from entailment. We showed that the class of terminating term rewriting systems is exactly learnable using entailment equivalence queries and request for hint queries. Particularly, polynomial time learning is possible for a class of TRS where the length of a derivation is bounded by an initial term. We also showed that a restricted subclass realizing top-down tree transductions, called k-variable linear tree translations, is efficiently learnable using entailment equivalence queries and entailment membership queries.

Most applications in semistructured data, such as Web and XML repositories, deal with labeled tree structures with unbounded branching [6]. Such applications also require the treatment of unordered trees. Unfortunately, our data model does not allow neither unbounded nor unordered trees. Therefore, extension for these structures is a future research problem.

References

1. D. Angluin, "Learning with hints," Proc. COLT'88, pp.223–237, 1988.
2. D. Angluin, "Queries and concept learning," Machine Learning, vol.2, pp.319–342, 1988.
3. H. Arimura, H. Ishizaka, T. Shinohara, S. Otsuki, "A generalization of the least general generalization," Machine Intelligence, vol.13, pp.59–85, Oxford University Press, 1994.
4. H. Arimura, "Learning Acyclic First-order Horn Sentences From Entailment," Proc. 7th Int. Workshop on Algorithmic Learning Theory (LNAI 1316), pp.432–445, 1997.
5. H. Arimura, H. Ishizaka, and T. Shinohara, "Learning unions of tree patterns using queries," Theoretical Computer Science, vol.185, pp.47–62, 1997.
6. J. Clark (ed.), "XSL Transformations (XSLT) Version 1.0," W3C Recommendation 19, `http://www.w3.org/TR/xslt`, November 1999.
7. N. Dershowitz, J.-P. Jouannaud, Rewrite Systems, Chapter 6, Formal Models and Semantics, Handbook of Theoretical Computer Science, Vol. B, Elseveir, 1990.
8. F. Drewes, Computation by Tree Transductions, Ph D. Thesis, University of Bremen, Department of Mathmatics and Informatics, February 1996.

9. M. Frazier and L. Pitt, Learning from entailment: an application to propositional Horn sentences, Proc. 10th Int. Conf. Machine Learning, pp.120–127, 1993.
10. M. Frazier, and L. Pitt, "CLASSIC learning," Proc. 7th COLT, pp.23–34, 1994.
11. R. Khardon, "Learning function-free Horn expressions," Proc. COLT'98 , pp.154–165, 1998.
12. C. D. Page and A. M. Frisch, "Generalization and learnability: a study of constrained atoms," Inductive Logic Programming, Academic Press, pp.l29–161, 1992.
13. G. D. Plotkin. "A note on inductive generalization," Machine Intelligence, vol.5, pp.153–163. Edinburgh University Press, 1970.
14. G. D. Plotkin. "A further note on inductive generalization," Machine Intelligence, vol.6, pp.101–124. Edinburgh University Press, 1971.
15. C. Reddy and P. Tadepalli, "Learning Horn definitions with equivalence and membership queries," Proc. the seventh ILP workshop (LNAI 1297), pp.243–255, 1997.
16. C. Reddy and P. Tadepalli, "Learning first-order acyclic Horn programs from entailment," Proc. Int. Conf. Machine Learning, 1998.
17. H. Sakamoto, H. Arimura, S. Arikawa, Identification of tree translation rules from examples, Proc. the 5th International Colloquium on Grammatical Inference (ICGI-2000), LNAI, Springer-Verlag, Sep. 2000. (To appear)
18. L. G. Valiant, "A theory of learnable," Comm. ACM, vol.27, pp.1134–1142, 1984.
19. A. Yamamoto, "Representing inductive inference with SOLD-resolution," "Proc. the IJCAI'97 Workshop on Abduction and Induction in AI, pp.59–63, 1997.
20. A. Yamamoto, "Which hypotheses can be found with inverse entailment?" Proc. the Seventh International Workshop on Inductive Logic Programming (LNAI 1297), pp.296–308, 1997. Also in Proc. the IJCAI'97 Workshop on Frontiers of Inductive Logic Programming, pp.19–23 , 1997.
21. A. Yamamoto, "An inference method for the complete inverse of relative subsumption," New Generation Computing, vol.17, no.1, pp.99–117, 1999.
22. A. Yamamoto and H. Arimura, "Inductive logic programming: from logic of discovery to machine learning," IEICE Trans. Inf. & Syst., E-83-D (1), pp.10–18, 2000.