Learnability of Term Rewrite Systems from Positive Examples

M.R.K. Krishna Rao

Information and Computer Science Department
King Fahd University of Petroleum and Minerals,
Dhahran 31261, Saudi Arabia.
Email: `krishna@ccse.kfupm.edu.sa`

Abstract

Learning from examples is an important characteristic feature of intelligence in both natural and artificial intelligent agents. In this paper, we study learnability of term rewriting systems from positive examples alone. We define a class of linear-bounded term rewriting systems that are inferable from positive examples. In linear-bounded term rewriting systems, nesting of defined symbols is allowed in right-hand sides, unlike the class of flat systems considered in Krishna Rao [8]. The class of linear-bounded TRSs is rich enough to include many divide-and-conquer programs like addition, logarithm, tree-count, list-count, split, append, reverse etc.

1 Introduction

Starting from the influential works of Gold [5] and Blum and Blum [3], a lot of effort has gone into developing a rich theory about inductive inference and the classes of concepts which can be learned from both positive (examples) and negative data (counterexamples) and the classes of concepts which can be learned from positive data alone. The study of inferability from positive data alone is important because negative information is hard to obtain in practice –positive examples are much easier to generate by conducting experiments than the negative examples in general. In his seminal paper [5] on inductive inference, Gold proved that even simple classes of concepts like the class of regular languages cannot be infered from positive examples alone. This strong negative result disappointed the scientists in the field until Angluin [1] has given a characterization of the classes of concepts that can be infered from positive data alone and exhibited a few nontrivial classes of concepts inferable from positive data. This influential paper inspired further research on the inductive inference from positive data. Since then many positive results are published about inductive inference of logic programs and pattern languages from positive data (see a.o., [9, 2, 10, 7, 8]. To the best of our knowledge, inductive inference of term rewriting systems from positive data has not received much attention – [8] is the only publication on this topic so far.

In the last few decades, term rewriting systems have played a fundamental role in the analysis and implementation of abstract data type specifications, decidability of word problems, computability theory, design of functional programming languages (e.g. Miranda), integration of functional and logic programming paradigms, and artificial intelligence – theorem proving and automated reasoning.

In this paper, we propose a class of linear-bounded term rewriting systems that are inferable from positive examples. Linear-bounded TRSs have a nice property that the size of redexes in an innermost derivation starting from a flat term t is bounded by the size of the initial term t. This property ensures that we only need to consider rewrite rules whose sides are bounded by the size of the examples in learning linear-bounded TRSs from positive data.

The class of linear-bounded TRSs is rich enough to include many divide-and-conquer programs like addition, logarithm, tree-count, list-count, split, append, reverse etc. The relation between the class of linear-bounded TRSs and the class of simple flat TRSs recently introduced in [8] is discussed in a later section. In particular, flat TRSs can define functions (like doubling), whose output is bigger in size than the input, which is not possible with linear-bounded TRSs. On the other hand, flat TRSs do not allow nesting of defined symbols in the rewrite rules, which means that we cannot define functions like reverse and quick-sort that can be defined by a linear-bounded TRS. Due to space restrictions, many proofs are omitted.

The rest of the paper is organized as follows. The next section gives preliminary definitions and results needed about inductive inference. In section 3, we define the class of linear-bounded TRSs and establish a few properties of them in section 4. The inferability of linear-bounded TRSs from positive data is established in section 5. The final section concludes with a discussion on open problems.

2 Preliminaries

We assume that the reader is familiar with the basic terminology of term rewriting and inductive inference and use the standard terminology from [6, 4] and [5, 9]. The alphabet of a first order language L is a tuple $\langle \Sigma, \mathcal{X} \rangle$ of mutually disjoint sets such that Σ is a finite set of function symbols and $\mathcal{X}$ is a set of variables.. In the following, $\mathcal{T}(\Sigma, \mathcal{X})$ denotes the set of terms constructed from the function symbols in Σ and the variables in $\mathcal{X}$. The size of a term t, denoted by $|t|$, is defined as the number of occurrences of symbols (except the punctuation symbols) occurring in it.

Definition 1 A *term rewriting system* (TRS, for short) $\mathcal{R}$ is a pair (Σ, R) consisting of a set Σ of function symbols and a set R of rewrite rules of the form $l \rightarrow r$ satisfying:
(a) l and r are first order terms in $\mathcal{T}(\Sigma, \mathcal{X})$,
(b) left-hand-side l is not a variable and
(c) each variable occuring in r also occurs in l.

Example 1 The following TRS defines multiplication over natural numbers.

$$\mathtt{a(0,y)} \rightarrow \mathtt{y}$$
$$\mathtt{a(s(x),y)} \rightarrow \mathtt{s(a(x,y))}$$

$$\mathtt{m(0,y)} \rightarrow \mathtt{0}$$
$$\mathtt{m(s(x),y)} \rightarrow \mathtt{a(y,m(x,y))}$$

Here, a stands for addition and m stands for multiplication. ◇

Definition 2 A *context* $C[,\ldots,]$ is a term in $\mathcal{T}(\Sigma \cup \{\diamond\}, \mathcal{X})$. If $C[,\ldots,]$ is a context containing n occurrences of $\diamond$ and $t_1,\ldots,t_n$ are terms then $C[t_1,\ldots,t_n]$ is the result of replacing the occurrences of $\diamond$ from left to right by $t_1,\ldots,t_n$. A context containing precisely 1 occurrence of $\diamond$ is denoted $C[\,]$.

Definition 3 The *rewrite relation* $\Rightarrow_{\mathcal{R}}$ induced by a TRS $\mathcal{R}$ is defined as follows: $s \Rightarrow_{\mathcal{R}} t$ if there is a rewrite rule $l \rightarrow r$ in $\mathcal{R}$, a substitution σ and a context $C[\,]$ such that $s \equiv C[l\sigma]$ and $t \equiv C[r\sigma]$. We say that s *reduces to* t in *one rewrite* (or reduction) *step* if $s \Rightarrow_{\mathcal{R}} t$ and say s *reduces to* t (or t is reachable from s) if $s \Rightarrow^*_{\mathcal{R}} t$, where $\Rightarrow^*_{\mathcal{R}}$ is the transitive-reflexive closure of $\Rightarrow_{\mathcal{R}}$). The subterm $l\sigma$ in s is called a *redex*. A redex is an *innermost redex* if no proper subterm of it is a redex. A derivation $s \Rightarrow^*_{\mathcal{R}} t$ is an *innermost derivation* if each reduction step in it reduces an innermost redex.

Example 2 The following innermost derivation shows a computation of the value of the term $\mathtt{m(s(s(0)),s(s(s(0))))}$ by the above *TRS*.

$$\begin{aligned}
&\mathtt{m(s(s(0)),s(s(s(0))))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),m(s(0),s(s(s(0)))))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),a(s(s(s(0))),m(0,s(s(s(0))))))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),a(s(s(s(0))),0))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),s(a(s(s(0)),0)))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),s(s(a(s(0),0))))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),s(s(s(a(0,0)))))}\\
&\quad\Rightarrow \mathtt{a(s(s(s(0))),s(s(s(0))))}\\
&\quad\Rightarrow \mathtt{s(a(s(s(0)),s(s(s(0)))))}\\
&\quad\Rightarrow \mathtt{s(s(a(s(0),s(s(s(0))))))}\\
&\quad\Rightarrow \mathtt{s(s(s(a(0,s(s(s(0)))))))}\\
&\quad\Rightarrow \mathtt{s(s(s(s(s(s(0))))))}
\end{aligned}$$

This is one of the many possible derivations from $\mathtt{m(s(s(0)),s(s(s(0))))}$. Since the system is both terminating and confluent, every derivation (innermost or not) from this term ends in the same final value $\mathtt{s(s(s(s(s(s(0))))))}$. ◇

Remark 1 The conditions (b) left-hand-side l is not a variable and (c) each variable occuring in r also occurs in l of Definition 1 avoid trivial nonterminating computations. If a rewrite rule $x \rightarrow r$ with a varible left-hand-side is present in a *TRS*, every term can be rewritten by this rule and hence no normal form exist resulting in nonterminating computations. If the right-hand-side r contains a variable y not present in the left-hand-side l of a rule $l \rightarrow r$ such that $r \equiv C[y]$, then the term l can be rewritten to $C[l]$ (substitution σ replacing the extra-variable by l) resulting in ever growing terms and obvious nontermination.

Definition 4 Let U and E be two recursively enumerable sets, whose elements are called *objects* and *expressions* respectively.

- A *concept* is a subset $\Gamma \subseteq U$.
- An *example* is a tuple $\langle A, a\rangle$ where $A \in U$ and $a = \mathtt{true}$ or $\mathtt{false}$. Example $\langle A, a\rangle$ is *positive* if $a = \mathtt{true}$ and *negative* otherwise.
- A concept Γ is *consistent* with a sequence of examples $\langle A_1, a_1\rangle, \ldots, \langle A_m, a_m\rangle$ when $A_i \in \Gamma$ if and only if $a_i = \mathtt{true}$, for each $i \in [1, m]$.
- A *formal system* is a finite subset $R \subseteq E$.
- A *semantic mapping* is a mapping Φ from formal systems to concepts.
- We say that a formal system R *defines* a concept Γ if $\Phi(R) = \Gamma$.

Definition 5 A *concept defining framework* is a triple $\langle U, E, \Phi\rangle$ of a universe U of objects, a set E of expressions and a semantic mapping Φ.

Definition 6 A class of concepts $C = \{\Gamma_1, \Gamma_2, \ldots\}$ is an *indexed family of recursive concepts* if there exists an algorithm that decides whether $w \in \Gamma_i$ for any object w and natural number i.

Here onwards, we fix a concept defining framework $\langle U, E, \Phi\rangle$ arbitrarily and only consider indexed families of recursive concepts.

Definition 7 A *positive presentation* of a nonempty concept $\Gamma \subseteq U$ is an infinite sequence $w_1, w_2, \ldots$ of objects (positive examples) such that $\{w_i \mid i \geq 1\} = \Gamma$.

An *inference machine* is an effective procedure that requests an object as an example from time to time and produces a concept (or a formal system defining a concept) as a conjecture from time to time. Given a positive presentation $\sigma = w_1, w_2, \ldots$, an inference machine IM generates a sequence of conjectures $g_1, g_2, \cdots$. We say that IM *converges to* g on input σ if the sequence of conjectures $g_1, g_2, \ldots$ is finite and ends in g or there exists a positive integer k_0 such that $g_k = g$ for all $k \geq k_0$.

Definition 8 A class C of concepts is *inferable from positive data* if there exists an inference machine IM such that for any $\Gamma \in C$ and any positive presentation σ of Γ, IM converges to a formal system g such that $\Phi(g) = \Gamma$.

We need the following result of Shinohara [9] in proving our result.

Definition 9 A semantic mapping Φ is *monotonic* if $R \subseteq R'$ implies $\Phi(R) \subseteq \Phi(R')$. A formal system R is *reduced w.r.t.* $S \subseteq U$ if $S \subseteq \Phi(R)$ and $S \not\subseteq \Phi(R')$ for any proper subset $R' \subset R$.

Definition 10 A concept defining framework $\mathcal{C} = \langle U, E, \Phi\rangle$ has *bounded finite thickness* if

1. Φ *is monotonic* and
2. *for any finite set* $S \subseteq U$ *and any* $m \geq 0$, *the set* $\{\Phi(R) \mid R$ *is reduced w.r.t.* S *and* $|R| \leq m\}$ *is finite.*

Theorem 1 (Shinohara [9])
If a concept defining framework $\mathcal{C} = \langle U, E, \Phi\rangle$ has bounded finite thickness, then *the class*

$$\mathcal{C}^m = \{\Phi(R) \mid R \subseteq E, |R| \leq m\}$$

of concepts is inferable from positive data for every $m \geq 1$.

3 Linear-bounded Term Rewriting Systems

In the following, we partition Σ into set D of defined symbols that may occur as the outermost symbol of left-hand-side of rules and set C of constructor symbols that do not occur as the outermost symbol of left-hand-side of rules.

Definition 11 The set $D_{\mathcal{R}}$ of *defined* symbols of a term rewriting system $\mathcal{R}(\mathcal{F}, R)$ is defined as $\{root(l) \mid l \to r \in R\}$ and the set $C_{\mathcal{R}}$ of *constructor* symbols of $\mathcal{R}(\mathcal{F}, R)$ is defined as $\mathcal{F} - D_{\mathcal{R}}$.

To show the defined and constructor symbols explicitly, we may write the above rewrite system as $\mathcal{R}(D_{\mathcal{R}}, C_{\mathcal{R}}, R)$ and omit the subscript when such omission does not cause any confusion. The terms containing no defined symbols are called constructor terms, and we refer to the terms of the form $f(t_1, \cdots, t_n)$ such that f is a defined symbol and $t_1, \ldots, t_n$ are constructor terms as level 1 terms. In this paper, we only consider constructor systems – left-hand sides are level 1 terms.

We need the following definition in the sequel.

Definition 12 An argument filter is a mapping π that assigns to every defined symbol of arity n, a list of argument positions $[i_1, \ldots, i_k]$ such that $1 \le i_1 < i_2 < \cdots, < i_k \le n$.

Unlike the usual practice in termination (and context sensitive rewriting) literature, we use argument filters only for defined symbols and do not distinguish the case of $\pi(f)$ being a single argument.

The following notion of parametric size over constructor terms and level 1 terms is central to our results.

Definition 13 For a constructor term t, the *parametric size* $[t]$ of t is defined recursively as follows:

- if t is a variable x then $[t]$ is a linear expression x,
- if t is a constant then $[t]$ is zero,
- if $t = f(t_1, \ldots, t_n)$ then $[t]$ is a linear expression $1 + [t_1] + \cdots + [t_n]$.

For a level 1 term $t \equiv f(t_1, \cdots, t_n)$, the *parametric size* $[t]$ of t is defined as $[t_{i_1}] + \cdots + [t_{i_k}]$ when $\pi(f) = [i_1, \ldots, i_k]$.

Example 3 The parametric sizes of constructor terms `a`, `h(a,x,b)`, `h(g(a), g(g(x)), g(y))` are 0, $x+1$, $5+x+y$ respectively. The parametric size of level 1 term `f(g(a), g(g(x)), g(y))` with argument filter $\pi(f) = [1, 3]$ is $[g(a)] + [g(y)] = 1 + 1 + y = 2 + y$. $\diamond$

The following function **LIgen** generates a set of equations and two sets of linear inequalities from a given rewrite rule $l \to r$ in a constructor system and an argument filter π (note that π is used by this function implicitly through Def. 13). It uses fresh variables $Var_1, Var_2, \cdots$ which do not occur in the rewrite system under consideration.

```
function LIgen($l \to r,\ \pi$);
begin
  $EQ := \phi$; $LI_1 := \phi$; $LI_2 := \phi$;
  $i := 0$;          /* counter for fresh variables. */
  while $r$ contains defined symbols do
  begin
    Let $r \equiv C[u_1, \ldots, u_m]$, showing all the level 1
                                      subterms of $r$;
    $r := C[Var_{i+1}, \ldots, Var_{i+m}]$;
    $EQ := EQ \cup \{Var_{i+1} = u_1, \ldots, Var_{i+m} = u_m\}$;
    for $j := 1$ to $m$ do
    begin
      $ineq1_{i+j} := [u_j] \ge Var_{i+j}$;
      $ineq2_{i+j} := [l] \ge [u_j]$
    end
    $LI_1 := LI_1 \cup \{ineq1_{i+1}, \ldots, ineq1_{i+m}\}$;
    $LI_2 := LI_2 \cup \{ineq2_{i+1}, \ldots, ineq2_{i+m}\}$;
    $i := i + m$
  end;
  $LI_2 := LI_2 \cup \{ineq2_0 : [l] \ge [r]\}$;
end;
```

The above function **LIgen** introduces one fresh variable (and one equation in EQ and one inequality each in LI_1 and LI_2) corresponding to each defined symbol in the right-hand side term r of the rule $l \to r$. If a defined symbol f occurs above another defined symbol g in r and variables Var_i and Var_j correspond to f and g respectively, then $i > j$. The set EQ of equations and the numbering of inequalities are only needed in the proofs in the sequel.

Now, we are in a position to define the class of linear-bounded TRSs.

Definition 14 Let $\mathcal{R}$ be a constructor system and π be an argument filter. Then, $\mathcal{R}$ is a linear-bounded system w.r.t. π if each rule in it is linear-bounded w.r.t. π. A rewrite $l \to r$ is linear-bounded w.r.t. π if the inequalities in LI_1 imply each inequality in LI_2, where LI_1 and LI_2 are the sets of inequalities generated by the function **LIgen**$(l \to r,\ \pi)$.

A constructor system is linear-bounded if it is linear-bounded w.r.t. some argument filter π.

Remark 2 The validity of (linear) inequalities is traditionally defined as the follows: *the inequality* **expression1** $\ge$ **expression2** *is valid if and only if it is valid for all possible assignments of values to variables in it.* In the sequel, we only talk of sizes which are obviously non-negative and hence *the inequality* **expression1** $\ge$ **expression2** *is valid if and only if it is valid for all possible assignments of non-negative values to variables in it.* According to this, $X+1 > X$ is valid but $X + Y > X$ is not valid because Y can take a zero value and $X + 0$ is not greater than X. Similarly, $2X > X$ is not valid because X can take a zero value. However, both $X + Y \ge X$ and $2X \ge X$ are valid.

The following examples illustrate the concept of linear-bounded systems. We use short notations $LI_1(l \to r)$ and $LI_2(l \to r)$ to denote the inequalities generated by **LIgen**$(l \to r,\ \pi)$, when π is clear from the context (and write LI_1 and LI_2 when the rule is also clear).

Example 4 Consider the following constructor system reversing a list.

$\mathtt{app(nil, y)} \to \mathtt{y}$
$\mathtt{app(cons(x, z), y)} \to \mathtt{cons(x, app(z, y))}$
$\mathtt{rev(nil)} \to \mathtt{nil}$
$\mathtt{rev(cons(x, z))} \to \mathtt{app(rev(z), cons(x, nil))}$

We show this system to be linear-bounded w.r.t. argument filter π such that $\pi(app) = [1, 2]$ and $\pi(rev) = [1]$. For the first rule, $LI_1 = \phi$ and $LI_2 = \{\mathtt{y} \ge \mathtt{y}\}$. Since $\mathtt{y} \ge \mathtt{y}$ is a valid inequality, LI_1 obviously implies LI_2 and hence this rule is linear-bounded. Similarly, the third rule can be easily shown to be linear-bounded (with $LI_1 = \phi$ and $LI_2 = \{\mathtt{0} \ge \mathtt{0}\}$).

For the second rule, $LI_1 = \{\mathtt{z} + \mathtt{y} \ge \mathtt{Var}_1\}$ and $LI_2 = \{\mathtt{x} + \mathtt{y} + \mathtt{z} + \mathtt{1} \ge \mathtt{z} + \mathtt{y},\ \mathtt{x} + \mathtt{y} + \mathtt{z} + \mathtt{1} \ge \mathtt{x} + \mathtt{Var}_1 + \mathtt{1}\}$. The first inequality $\mathtt{x}+\mathtt{y}+\mathtt{z}+\mathtt{1} \ge \mathtt{z}+\mathtt{y}$

in LI_2 is a valid inequality and the second inequality $\mathtt{x}+\mathtt{y}+\mathtt{z}+1 \geq \mathtt{x}+\mathtt{Var}_1+1$ in LI_2 is implied by the inequality $\mathtt{z}+\mathtt{y} \geq \mathtt{Var}_1$ in LI_1. Therefore, this rule is linear-bounded.

For the fourth rule, $LI_1 = \{\mathtt{z} \geq \mathtt{Var}_1,\ \mathtt{Var}_1+\mathtt{x}+1 \geq \mathtt{Var}_2\}$ and $LI_2 = \{\mathtt{x}+\mathtt{z}+1 \geq \mathtt{z},\ \mathtt{x}+\mathtt{z}+1 \geq \mathtt{Var}_1+\mathtt{x}+1,\ \mathtt{x}+\mathtt{z}+1 \geq \mathtt{Var}_2\}$. The first inequality $\mathtt{x}+\mathtt{z}+1 \geq \mathtt{z}$ in LI_2 is a valid inequality, the second inequality $\mathtt{x}+\mathtt{z}+1 \geq \mathtt{Var}_1+\mathtt{x}+1$ in LI_2 is implied by the inequality $\mathtt{z} \geq \mathtt{Var}_1$ in LI_1, and the third inequality $\mathtt{x}+\mathtt{z}+1 \geq \mathtt{Var}_2$ in LI_2 is implied by the two inequalities $\mathtt{z} \geq \mathtt{Var}_1$ and $\mathtt{Var}_1+\mathtt{x}+1 \geq \mathtt{Var}_2$ in LI_1. Therefore, this rule is linear-bounded too. $\diamond$

4 Some Properties of Linear-bounded Systems

In this section, we prove some properties of linear-bounded systems. A nice property of the class of linear-bounded systems is that it is decidable whether a given TRS is linear-bounded or not, as this problem can be reduced to the satisfiability problem of linear inequalities.

Theorem 2 *It is decidable whether a TRS $\mathcal{R}$ is linear-bounded or not.*

The following theorem captures the basic idea of linear-bounded systems – the size of output is bounded by the size of input.

Theorem 3 Let $\mathcal{R}$ be a linear-bounded TRS and t be a level 1 term with root in D. If $t \Rightarrow^* v$ is an innermost derivation and v is a constructor term (i.e., a normal form), then the parametric sizes of t and v satisfy the property $[t] \geq [v]$.

Further, the size of any innermost redex in the above derivation is bounded by the size of the initial term.

Theorem 4 Let $\mathcal{R}$ be a linear-bounded TRS and t be a level 1 term with root in D. If $t \Rightarrow^* u$ is an innermost derivation such that w is an innermost redex in u, then the parametric sizes of t and w satisfy the property $[t] \geq [w]$.

The above characteristic properties of linear-bounded *TRS*s ensure that it is decidable whether a flat term t reduces to a constructor term u by a linear-bounded system or not.

Theorem 5 *If t is a level 1 term with root in D, u is a constructor term and R is a linear-bounded TRS, it is decidable whether $t \Rightarrow^*_R u$ or not.*

5 Inferability of linear-bounded TRSs from Positive Data

In this section, we establish inductive inferability of linear-bounded TRSs from positive data.

Definition 15 Let LB_k be the set of all linear-bounded rules of the form $l \rightarrow r$ such that $|l|+|r| \leq k$, FC be the cartesian product of (a) the set of all level 1 terms with root in D and (b) the set of all constructor terms, and Φ be a semantic mapping such that $\Phi(R)$ is the relation $\{(s,t) \mid s \Rightarrow^*_R t,\ s$ is a level 1 term with root in D and t is a constructor term$\}$. The concept defining framework $\langle LB_k, FC, \Phi\rangle$ is denoted by $\mathbf{LBF_k}$.

The following Lemma follows from Theorems 5 and 2.

Lemma 1 The class of rewrite relations defined by linear-bounded TRSs is an indexed family of recursive concepts.

The following theorem plays the predominant role in proving our main result.

Theorem 6 *The concept defining framework* $\mathbf{LBF_k} = \langle LB_k, FC, \Phi\rangle$ *has bounded finite thickness.*

Proof: Since Φ is the rewrite relation, it is obviously monotonic, i.e., $\Phi(R_1) \subseteq \Phi(R_2)$ whenever $R_1 \subseteq R_2$.

Consider a finite relation $S \subseteq FC$ and a TRS $R \subseteq LB_k$ containing at most $m \geq 1$ rules such that R is reduced w.r.t. S. Let n be an integer such that $n \geq [t]$ for every $(t,u) \in S\}$. Let S' be the set of innermost redexes in innermost derivations $t \Rightarrow^* u$ such that $(t,u) \in S\}$. By Theorem 4, $n \geq [w]$ for every term $w \in S'$. Since R is reduced w.r.t. S, every rule in R is used in derivations of S. Hence, $n \geq [l]$ for every rule $l \rightarrow r \in R$.

Since Σ is finite, there are only finitely many linear-bounded TRSs containing at most m rules of the form $l \rightarrow r$ such that $n \geq [l]$ and $|l|+|r| \leq k$ (except for the renaming of variables)[1]. Therefore, the set $\{\Phi(R) \mid R$ is reduced w.r.t. S and contains at most m rules$\}$ is finite. Hence, the concept defining framework $\langle LB_k, FC, \Phi\rangle$ has bounded finite thickness. $\diamond$

From this Theorem, Lemma 1 and Theorem 1, we obtain our main result.

Theorem 7 *For every $m, k \geq 1$, the class of linear-bounded TRSs with at most m rules of size at most k is inferable from positive data.*

6 Discussion

In this paper, we study inductive inference of term rewriting systems from positive data. A class of linear-bounded TRSs *inferable from positive data* is defined. This class of TRSs is rich enough to include divide-and-conquer programs like addition, logarithm, tree-count, list-count, split, append, reverse etc. To the best of our knowledge, only known results about inductive inference of term rewriting systems from positive data are from [8], where the class of simple flat[2] TRSs is shown to be inferable from positive data. The classes of simple flat TRSs and linear-bounded TRSs are incomparable for the following reasons.

1. Linear-bounded TRSs only capture functions whose output is bounded by the size of the inputs. Functions like addition, list-count, split, append, reverse have such a property. But functions like multiplication and doubling are beyond linear-bounded TRSs as the size of their output is bigger than that of the input. The following simple flat TRS computes the double of a given list (output contains each element of the input twice as often).

```
double(nil) → nil
double(cons(H,T)) → cons(H,cons(H,double(T)))
```

[1]Since argument filters –and hince the parametric linear– ignore some arguments, the additonal condition $|l|+|r| \leq k$ is needed. In [8], no argument filters are used and hence this additional condition is not needed.

[2]A TRS is simple flat if defined symbols are not nested in any rule and the sum of the sizes of arguments of defined symbols in the right-hand side of a rule is bounded by the sum of the sizes of arguments of defined symbols in the left-hand side [8].

This shows that there are functions that can be computed by simple flat TRSs but not by linear-bounded TRSs.

2. The rewrite system for computing reverse of a list given in Example 4 is linear-bounded, but it is beyond simple flat TRSs as it involves nesting of defined symbols. This shows that there are functions that can be computed by linear-bounded but not by simple flat TRSs.

Open problem: In view of this incomparability of the simple flat TRSs and linear-bounded TRSs, it will be very useful to work towards extending the frontiers of inferable classes of rewrite systems and characterize some classes of TRSs having the expressive power of both simple flat TRSs and linear-bounded TRSs, and yet inferable from positive data.

Acknowledgements: The author would like to thank the King Fahd University of Petroleum and Minerals for the generous support provided by it in conducting this research.

References

[1] D. Angluin (1980), *Inductive inference of formal languages from positive data*, Information and Control **45**, pp. 117-135.

[2] 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, pp. 365-375, IOS press.

[3] L. Blum and M. Blum (1975), *Towards a mathematical theory of inductive inference*, Information and Control **28**, pp. 125-155.

[4] N. Dershowitz and J.-P. Jouannaud (1990), *Rewrite Systems*, In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B, pp. 243-320, North-Holland.

[5] E.M. Gold (1967), *Language identification in the limit*, Information and Control **10**, pp. 447-474.

[6] J.W. Klop (1992), *Term Rewriting Systems*, in S. Abramsky, D. Gabby and T. Maibaum (ed.), *Handbook of Logic in Computer Science*, Vol. 1, Oxford Press, 1992.

[7] M. R. K. Krishna Rao (2000), *Some classes of prolog programs inferable from positive data*, Theor. Comput. Sci. **241**, pp. 211-234.

[8] M. R. K. Krishna Rao (2004), *Inductive Inference of Term Rewriting Systems from Positive Data*, in Proc. of Algorithmic Learning Theory, ALT'04, Lecture Notes in Artificial Intelligence **3244**, pp. 69-82, Springer-Verlag.

[9] T. Shinohara (1991), *Inductive inference of monotonic formal systems from positive data*, New Generation Computing **8**, pp. 371-384.

[10] Takeshi Shinohara, Hiroki Arimura (2000), *Inductive inference of unbounded unions of pattern languages from positive data*, Theor. Comput. Sci. **241**, pp. 191-209.

[11] Y. Toyama (1987), *Counterexamples to termination for the direct sum of term rewriting systems*, Inf. Process. Lett. **25**, pp. 141-143.