Algorithms for Learning Function Distinguishable Regular Languages

Henning Fernau[1]* and Agnes Radl[2]

[1] School of Electrical Engineering and Computer Science, University of Newcastle
University Drive, NSW 2308 Callaghan, Australia
fernau@cs.newcastle.edu.au
[2] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
agnes.radl@student.uni-tuebingen.de

Abstract. Function distinguishable languages were introduced as a new methodology of defining characterizable subclasses of the regular languages which are learnable from text. Here, we give details on the implementation and the analysis of the corresponding learning algorithms. We also discuss problems which might occur in practical applications.

1 Introduction

Identification in the limit from positive samples, also known as *exact learning from text* as proposed by Gold [10], is one of the oldest yet most important models of grammatical inference. Since not all regular languages can be learned exactly from text, the characterization of *identifiable* subclasses of regular languages is a useful line of research, because the regular languages are a very basic language family, see also the discussions in [12] regarding the importance of finding *characterizable* learnable language classes.

In [4], we introduced the so-called function-distinguishable languages as a rich source of examples of identifiable language families. Among the language families which turn out to be special cases of our approach are the k-reversible languages [1] and (reversals of) the terminal-distinguishable languages [13,14], which belong, according to Gregor [11], to the most popular identifiable regular language classes. Moreover, we have shown [4] how to transfer the ideas underlying the well-known identifiable language classes of k-testable languages, k-piecewise testable languages and threshold testable languages to our setting. In a nutshell, an identification algorithm for f-distinguishable languages assigns to every finite set of samples $I_+ \subseteq T^*$ the smallest f-distinguishable language containing I_+ by subsequently merging states which cause conflicts to the definition of f-distinguishable automata, starting with the simple prefix tree automaton accepting I_+.

* Work was done while the author was with Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany

T. Caelli et al. (Eds.): SSPR&SPR 2002, LNCS 2396, pp. 64–73, 2002.

Another interesting property of each class f-DL of function distinguishable languages is established in [6]: the approximability of the whole class of regular languages in the sense that, given any regular language L, a learning algorithm for f-DL infers, given L, the smallest language $L' \in f$-DL including L.

Applications of the learnability of function-distinguishable languages have been reported in [8] for the identifiability of parallel communicating grammar systems and in [7] for inferring document type definitions of XML documents.

Here, we aim at giving more details on the implementation and analysis of the learning algorithms for function-distinguishable languages. We also give a proof of a counterexample originally given by Radhakrishnan and Nagaraja.

2 General Definitions

Σ^* is the set of words over the alphabet Σ. Σ^k ($\Sigma^{<k}$) collects the words whose lengths are equal to (less than) k. λ denotes the empty word. $\mathrm{Pref}(L)$ is the set of prefixes of L and $u^{-1}L = \{ v \in \Sigma^* | uv \in L \}$ is the quotient of $L \subseteq \Sigma^*$ by u.

Regular languages can be characterized by (deterministic) finite automata $A = (Q, T, \delta, q_0, Q_F)$, where Q is the state set, $\delta \subseteq Q \times T \times Q$ is the transition relation, $q_0 \in Q$ is the initial state and $Q_F \subseteq Q$ is the set of final states. As usual, δ^* denotes the extension of the transition relation to arbitrarily long input words. The language defined by an automaton A is written $L(A)$. An automaton is called *stripped* iff all states are accessible from the initial state and all states lead to some final state. Observe that the transition function of a stripped deterministic finite automaton is not total in general.

We denote the minimal deterministic automaton of the regular language L by $A(L)$. Recall that $A(L) = (Q, T, \delta, q_0, Q_F)$ can be described as follows: $Q = \{u^{-1}L | u \in \mathrm{Pref}(L)\}$, $q_0 = \lambda^{-1}L = L$; $Q_F = \{u^{-1}L | u \in L\}$; and $\delta(u^{-1}L, a) = (ua)^{-1}L$ with $u, ua \in \mathrm{Pref}(L)$, $a \in T$. According to our definition, any minimal deterministic automaton is stripped.

The *product automaton* $A = A_1 \times A_2$ of two automata $A_i = (Q_i, T, \delta_i, q_{0,i}, Q_{F,i})$ for $i = 1, 2$ is defined as $A = (Q, T, \delta, q_0, Q_F)$ with $Q = Q_1 \times Q_2$, $q_0 = (q_{0,1}, q_{0,2})$, $Q_F = Q_{F,1} \times Q_{F,2}$, $((q_1, q_2), a, (q_1', q_2')) \in \delta$ iff $(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$.

3 Function Distinguishable Languages

In order to avoid cumbersome case discussions, let us fix now T as the input alphabet of the finite automata we are going to discuss.

Definition 1. *Let F be some finite set. A mapping $f : T^* \to F$ is called a* distinguishing function *if $f(w) = f(z)$ implies $f(wu) = f(zu)$ for all $u, w, z \in T^*$.*

In the literature, we can find the terminal function [14]

$$\mathrm{Ter}(x) = \{ a \in T \mid \exists u, v \in T^* : uav = x \}$$

and, more generally, the k-terminal function [5]

$$\mathrm{Ter}_k(x) = (\pi_k(x), \mu_k(x), \sigma_k(x)), \quad \text{where}$$
$$\mu_k(x) = \{\, a \in T^{k+1} \mid \exists u, v \in T^* : uav = x \,\}$$

and $\pi_k(x)$ [$\sigma_k(x)$] is the prefix [suffix] of length k of x if $x \notin T^{<k}$, and $\pi_k(x) = \sigma_k(x) = x$ if $x \in T^{<k}$. The example $f(x) = \sigma_k(x)$ leads to the k-reversible languages, confer [1,5]. In particular, the trivial distinguishing function, whose range is a singleton set, characterizes the 0-reversible languages.

To every distinguishing function f, a finite automaton $A_f = (F, T, \delta_f, f(\lambda), F)$ can be associated by setting $\delta_f(q, a) = f(wa)$, where $w \in f^{-1}(q)$ can be chosen arbitrarily, since f is a distinguishing function. Here, we will formally introduce function distinguishable languages and discuss some formal language properties.

Definition 2. *Let $A = (Q, T, \delta, q_0, Q_F)$ be a finite automaton. Let $f : T^* \to F$ be a distinguishing function. A is called* f-distinguishable *if:*

1. *A is deterministic.*
2. *For all states $q \in Q$ and all $x, y \in T^*$ with $\delta^*(q_0, x) = \delta^*(q_0, y) = q$, we have $f(x) = f(y)$.*
   (In other words, for $q \in Q$, $f(q) := f(x)$ for some x with $\delta^*(q_0, x) = q$ is well-defined.)
3. *For all $q_1, q_2 \in Q$, $q_1 \neq q_2$, with either (a) $q_1, q_2 \in Q_F$ or (b) there exist $q_3 \in Q$ and $a \in T$ with $\delta(q_1, a) = \delta(q_2, a) = q_3$, we have $f(q_1) \neq f(q_2)$.*

Informally speaking, an automaton is f-distinguishable if backward nondeterminism conflicts can be resolved with the help of the distinguishing function f.

A language is *f-distinguishable* iff it can be accepted by an f-distinguishable automaton. The class of f-distinguishable languages is denoted by f-DL.

We need a suitable notion of a canonical automaton in the following.

Definition 3. *Let $f : T^* \to F$ be a distinguishing function and let $L \subseteq T^*$ be a regular set. Let $A(L, f)$ be the stripped subautomaton of the product automaton $A(L) \times A_f$. $A(L, f)$ is called* f-canonical automaton *of L.*

Theorem 1. *Let $f : T^* \to F$ be some distinguishing function. Then, $L \subseteq T^*$ is f-distinguishable iff $A(L, f)$ is f-distinguishable.* □

This characterization was proved in [4] and used in order to establish the inferability of f-DL. $A(L, f)$ was employed to construct a characteristic sample for L (with respect to f), and moreover, the $A(L, f)$ (note that $A(L, f)$ is usually larger than $A(L)$) are the hypothesis space of the learning algorithm.

4 An Extended Example

Radhakrishnan showed [13, Example 3.4] that the language L described by $ba^*c + d(aa)^*c$ lies in Ter-DL but its reversal does not. Consider the deterministic (minimal) automaton $A(L)$ with transition function δ (see Table 1). Is $A(L)$

Table 1. The transition functions δ, δ_{Ter} and $\delta_{\mathrm{inferred}}$

	a	b	c	d
$\to 0$	–	1	–	2
1	1	–	3	–
2	4	–	3	–
$3 \to$	–	–	–	–
4	2	–	–	–

Ter		a	b	c	d
$\emptyset$	$\to 0$	–	1	–	2
$\{b\}$	1	$1'$	–	3	–
$\{a,b\}$	$1'$	$1'$	–	$3'$	–
$\{d\}$	2	4	–	$3''$	–
$\{a,d\}$	$2'$	4	–	$3'''$	–
$\{b,c\}$	$3 \to$	–	–	–	–
$\{a,b,c\}$	$3' \to$	–	–	–	–
$\{c,d\}$	$3'' \to$	–	–	–	–
$\{a,c,d\}$	$3''' \to$	–	–	–	–
$\{a,d\}$	4	$2'$	–	–	–

Ter		a	b	c
$\emptyset$	$\to 0$	–	1	–
$\{b\}$	1	$1'$	–	3
$\{a,b\}$	$1'$	$1'$	–	$3'$
$\{b,c\}$	$3 \to$	–	–	–
$\{a,b,c\}$	$3' \to$	–	–	–

Ter-distinguishable? We have still to check whether it is possible to resolve the backward nondeterminism conflicts (the state 3 occurs two times in the column labelled c). This resolution possibility is formalized in the second and third condition in Definition 2. As to the second condition, the question is whether it is possible to assign Ter-values to states of $A(L)$ in a well-defined manner: assigning $\mathrm{Ter}(0) = \emptyset$ and $\mathrm{Ter}(4) = \{a,d\}$ is possible, but should we set $\mathrm{Ter}(1) = \{b\}$ (since $\delta^*(0,b) = 1$) or $\mathrm{Ter}(1) = \{a,b\}$ (since $\delta^*(0,ba) = 1$)?; similar problems occur with states 2 and 3. Let us therefore try another automaton accepting L, whose transition function δ_{Ter} is given by Table 1, we indicate the Ter-values of the states in the first column of the table. As the reader may verify, δ_{Ter} basically is the transition table of the stripped subautomaton of the product automaton $A(L) \times A_{\mathrm{Ter}}$. One source of backward nondeterminism may arise from multiple final states, see condition 3.(a) of Def. 2. Since the Ter-values of all four finite states are different, this sort of nondeterminism can be resolved. Let us consider possible violations of condition 3.(b) of Def. 2. In the column labelled a, we find multiple occurrences of the same state entry:

- $\delta_{\mathrm{Ter}}(1,a) = \delta_{\mathrm{Ter}}(1',a) = 1'$: since $\mathrm{Ter}(1) = \{b\} \neq \mathrm{Ter}(1') = \{a,b\}$, this conflict is resolvable.
- $\delta_{\mathrm{Ter}}(2,a) = \delta_{\mathrm{Ter}}(2',a) = 4$: since $\mathrm{Ter}(2) = \{d\} \neq \mathrm{Ter}(2') = \{a,d\}$, this conflict is resolvable.

Observe that the distinguishing function f can be also used to design efficient backward parsing algorithms for languages in f-DL. The only thing one has to know are the f-values of all prefixes of the word w to be parsed. Let us try to check that $daac$ belongs to the language L in a backward fashion. For the prefixes, we compute: $\mathrm{Ter}(d) = \{d\}$, $\mathrm{Ter}(da) = \mathrm{Ter}(daa) = \{a,d\}$, and $\mathrm{Ter}(daac) = \{a,c,d\}$. Since $\mathrm{Ter}(w_1) = \{a,c,d\}$, we have to start our backward parse in state $3'''$. The column labelled c reveals that after reading the last letter c, we are in state $2'$. After reading the penultimate letter a, we are therefore in state 4. Reading the second letter a brings us into state 2, since the Ter-value

Table 2. The transition functions of A_{L^R} and of $A(L^R, \mathrm{Ter})$; X is one of $\{a, b, c\}$, $\{a, c, d\}$, $\{b, c\}$ and $\{c, d\}$

	a	b	c	d
$\to 0$	–	–	1	–
1	2	3	–	3
2	1	3	–	–
$3 \to$	–	–	–	–

	a	b	c	d
$\to (0, \{\lambda\})$	–	–	$(1, \{c\})$	–
$(1, \{c\})$	$(2, \{a, c\})$	$(3, \{b, c\})$	–	$(3, \{c, d\})$
$(1, \{a, c\})$	$(2, \{a, c\})$	$(3, \{a, b, c\})$	–	$(3, \{a, c, d\})$
$(2, \{a, c\})$	$(1, \{a, c\})$	$(3, \{a, b, c\})$	–	–
$(3, X) \to$	–	–	–	–

of the prefix left to be read is $\{d\} = \mathrm{Ter}(2)$. Finally, reading d brings us to the initial state 0; hence, *daac* is accepted by the automaton.

Let us discuss why L^R described by $ca^*b + c(aa)^*d$ is *not* in Ter-DL, as already Radhakrishnan claimed (without proof) [13, Example 3.4]. Table 2 shows the transition function of the minimal deterministic automaton A_{L^R} and the transition function of $A(L^R, \mathrm{Ter})$. As the reader may verify, $A(L^R, \mathrm{Ter})$ is *not* Ter-distinguishable. Our characterization theorem implies that L^R is not Ter-distinguishable either. A similar argument shows that L^R is not σ_1-distinguishable. On the contrary, L^R is σ_2-distinguishable. This can be seen by looking at $A(L^R, \sigma_2)$.

5 Inference Algorithm

We present an algorithm which receives an input sample set $I_+ = \{w_1, \ldots, w_M\}$ (a finite subset of the language $L \in f\text{-DL}$ to be identified) and finds the smallest language $L' \in f\text{-DL}$ which contains I_+.

The *prefix tree acceptor* $PTA(I_+) = (Q, T, \delta, q_0, Q_F)$ of a finite sample set $I_+ = \{w_1, \ldots, w_M\} \subset T^*$ is a deterministic finite automaton which is defined as follows: $Q = \mathrm{Pref}(I_+)$, $q_0 = \lambda$, $Q_F = I_+$ and $\delta(v, a) = va$ for $va \in \mathrm{Pref}(I_+)$.

A simple merging state inference algorithm `f-Ident` for f-DL now starts with the automaton $A_0 = PTA(I_+)$ and merges two arbitrarily chosen states q and q' which cause a conflict to the first or the third of the requirements for f-distinguishing automata.[1] This yields an automaton A_1. Again, choose two conflicting states p, p' and merge them to obtain an automaton A_2 and so forth, until one comes to an automaton A_t which is f-distinguishable. In this way, we get a chain of automata $A_0, A_1, \ldots, A_t$. Observe that each A_i is stripped, since A_0 is stripped. In a fashion analogous to the algorithm ZR designed by Angluin for inferring 0-reversible languages, a description of the algorithm `f-Ident`, where $f : T^* \to F$, can be given as follows:

Algorithm 1 (`f-Ident`*).*
Input: a nonempty positive sample $I_+ \subseteq T^*$.

[1] One can show that the second requirement won't ever be violated when starting the merging process with A_0 which trivially satisfies that condition.

Output: $A(L, f)$, where L is the smallest f-distinguishable language containing I_+.
*** Initialization
Let $A_0 = (Q_0, T, \delta_0, q_{0,0}, Q_{F,0}) = PTA(I_+)$.
For each $q \in Q_0$, compute $f(q)$.
Let π_0 be the trivial partition of Q_0.
Initialize the successor function s by defining $s(\{q\}, a) := \delta_0(q, a)$ for $q \in Q_0$, $a \in T$.[2]
Initialize the predecessor function p by $p(\{q\}, a) := (q', f(q'))$, with $\delta_0(q', a) := q$.[3]
Let LIST contain all pairs $\{q, q'\} \subseteq Q_0$ with $q \neq q'$, $q, q' \in Q_{F,0}$ and $f(q) = f(q')$.
Let $i := 0$.
*** Merging
While LIST$\neq \emptyset$ do begin
    Remove some element $\{q_1, q_2\}$ from LIST.
    Consider the blocks $B_1 = B(q_1, \pi_i)$ and $B_2 = B(q_2, \pi_i)$.
    If $B_1 \neq B_2$, then begin
        Let π_{i+1} be π_i with B_1 and B_2 merged.
        For each $a \in T$, do begin
            If both $s(B_1, a)$ and $s(B_2, a)$ are defined and not equal,
                then place $\{s(B_1, a), s(B_2, a)\}$ on LIST.
            If $s(B_1, a)$ is defined, then set $s(B_1 \cup B_2, a) := s(B_1, a)$;
            otherwise, set $s(B_1 \cup B_2, a) := s(B_2, a)$.
            For each $z \in F$, do begin
                If there are $(p_i, z) \in p(B_i, a)$, $i = 1, 2$, then:
                    If $B(p_1, \pi_i) \neq B(p_2, \pi_i)$, then place $\{p_1, p_2\}$ on LIST.
                    Set $p(B_1 \cup B_2, a) := p(B_1, a)$.
                If $(p_2, z) \in p(B_2, a)$ then:
                    If there is no p_1 with $(p_1, z) \in p(B_1, a)$,
                        then add (p_2, z) to $p(B_1 \cup B_2, a)$.
            end *** for z
        end *** for a
        Increment i by one.
        If $i = |Q_0| - 1$, then LIST$= \emptyset$.
    end *** if
end *** while

An induction shows that any pair $\{q, q'\}$ ever placed on LIST obeys $f(q) = f(q')$.

Example 1. Let us illustrate the work of `f-Ident` by means of an example (with f = Ter): Consider $I_+ = \{bc, bac, baac\}$. Since Ter(bac) = Ter($baac$), initially (only) the state pair $\{bac, baac\}$ of the PTA is placed onto LIST. In the first pass through the while-loop, these two states are merged. Since both $s(\{bac\}, u)$ and $s(\{baac\}, u)$ are undefined for any letter u, $s(\{bac, baac\}, u)$ will be also undefined. Since Ter(ba) = Ter(baa), the pair $\{ba, baa\}$ is placed on LIST when investigating the predecessors. In the next pass through the while-loop, $\{ba\}$ and $\{baa\}$ are merged, but no further mergeable pairs are created,

[2] According to the initialization, both s and p may be undefined for some arguments.
[3] Note that this state q' is uniquely defined in $PTA(I_+)$.

since in particular, the predecessors b and ba of ba and baa, respectively, have different Ter-values. Hence, the third transition function δ_{inferred} of Table 1 is inferred; for clarity, we indicate the Ter-values of the states in the first column of the table.

The somewhat peculiar names of the states were chosen in order to make the comparison with the Ter-distinguishable automaton presented in Section 4 easier for the reader. In terms of the block notion used in the inference algorithm, we have $0 = \{\lambda\}$, $1 = \{b\}$, $1' = \{ba, baa\}$, $3 = \{bc\}$, and $3' = \{bac, baac\}$. Observe that the resulting automaton is not the minimal automaton of the obtained language ba^*c, which is obtainable by merging state 1 with $1'$ and state 3 with $3'$.

Theorem 2 (Correctness, see [4]). *If $L \in f$-DL is enumerated as input to the algorithm* `f-Ident`*, it converges to the f-canonical automaton $A(L, f)$.* □

Here, we will give a detailed complexity analysis valid for the popular RAM computation model where arbitrarily large integers fit into one register or memory unit. This means that values of the s and p functions can be compared in unit time. The following analysis is based on an implementation which uses the operations UNION (of two disjoint subsets, i.e., classes, of a given n-element universe) and FIND (the class to which a given element belongs).[4]

Theorem 3 (Time complexity). *By using a standard union-find algorithm, the algorithm* `f-Ident` *can be implemented to run in time*

$$O(\alpha(2(|F|+1)(|T|+1)n, n)(|F|+1)(|T|+1)n),$$

where α is the inverse Ackermann function[5] *and n is the total length of all words in I_+ from language L, when L is the language presented to the learner for f-DL.*

Proof. In any case, $PTA(I_+)$ has basically n states; these states comprise the universe of the union-find algorithm. UNION will be applied no more than $n-1$ times, since then the inferred automaton will be trivial,

How many FIND operation will be triggered? Two FIND operations will be needed to compute the blocks B_1 and B_2 to which a pair (q_1, q_2) taken from LIST belongs. Apart from the initialization, a certain number of new elements is put onto LIST each time a UNION operation is performed. More precisely, each letter $a \in T$ may cause $\{s(B_1, a), s(B_2, a)\}$, as well as $|F|$ "predecessor pairs" $\{p_1, p_2\}$, to be put onto LIST. In the initialization phase, no more than $\min\{|F|^2, n^2\}$ elements are put onto LIST. So, no more than

$$(|F|+1)|T|(n-1) + \min\{|F|^2, n^2\} \leq (|F|+1)(|T|+1)n$$

elements are ever put onto LIST. □

[4] A thorough analysis of various algorithms is contained in [15]. A simplified analysis can be found in [3]. Angluin's algorithm ZR for 0-reversible languages works similarly.

[5] For the exact definition of α, we refer to [15].

Observe that this basically leads to an $O(\alpha(|T|^{2k+1}n, n)|T|^{2k+1}n)$ algorithm for k-reversible languages; but note that we output a different type of canonical automata compared with Angluin. When k is small compared to n (as it would be in realistic applications, where k could be considered even as a fixed parameter), our algorithm for k-reversible language inference would run in nearly linear time, since the inverse Ackermann function is an extremely slowly growing function, while Angluin [1] proposed a cubic learning algorithm (always outputting the minimal deterministic automaton). Similar considerations are also true for the more involved case of regular tree languages [9].

Note that the performance of `f-Ident` depends on the size of A_f (since the characteristic sample $\chi(L, f)$ we defined above depends on this size) and is in this sense "scalable", since "larger" A_f permit larger language families to be identified. More precisely, we can show:

Theorem 4. *Let f and g be distinguishing functions. If A_f is a homomorphic image of A_g, then f-DL $\subseteq$ g-DL.* □

This scalability leads to the natural question which distinguishing function one has to choose in one's application.

Let us assume that the user knows several "typical" languages $L_1, \ldots, L_r$. Possibly, the choice of $f_{L_1} \times \cdots \times f_{L_r}$ as distinguishing function has a range which is too large for practical implementation. Recall that the identification algorithm proposed in [4] exponentially depends on the size of the range of the distinguishing function. Therefore, the following problem is of interest:
Problem: Given $L_1, \ldots, L_r$, find a distinguishing function f with minimal range such that $L_1, \ldots, L_r$ lie all within f-DL. Although we expect this problem to be NP-hard, we have yet no proof. We even suspect the problem is hard if $r = 1$.

6 A Retrievable Implementation

Under `www-fs.informatik.uni-tuebingen.de/~fernau/GI.htm`, the program **inference** written in C++ can be retrieved. Usage:
inference -a [**-k**|**-t**|**-k**n|**-t**n] [*fdist*] *sample*

You can specify your own distinguishing function in the file *fdist* as the transition relation of a finite automaton. The format of *fdist* is a table where the rows are terminated by line feeds or by carriage returns and line feeds. Columns are separated by space characters or tabs or both. *fdist* must contain all the symbols used in *sample*. The learner generalizes according to the specified algorithm and writes the inferred automaton to standard output.

If the **-k** or **-t** option is given, *fdist* will be ignored. The **-k** option invokes a k-reversible inference algorithm. If **k** is followed by a positive integer n, σ_n will be used as the distinguishing function, else σ_0 will be used. The **-t** option causes the learner to use Ter as the distinguishing function—basically corresponding to the terminal distinguishable languages. If **-t** is followed by a positive integer n, Ter_n will be used as the distinguishing function. As an example, consider

invoking `inference -a -t sampleRN` with the samples *bc*, *bac*, *dc*, *daac*, *baac* and *daaaac* listed in the file `sampleRN`; this yields δ_{Ter} from Table 1.

Moreover, the program can be used to infer document type definitions (DTD) for XML documents base on the inference of function distinguishable languages, as explained in [7]. Usage:
inference -x [**-k**|**-t**|**-k***n*|**-t***n*] *xml-doc* [*dtd-file*]
Most of the options are used as explained before. Additionally, the created DTD rules will be written to the file *dtd-file*, if specified. An existing file *dtd-file* will be overwritten. Note that the DTD will most probably be incomplete because no attribute rules are inferred.

As a peculiarity, let us finally mention the 1-unambiguity requirement for DTDs: If some rule violates this requirement, the DTD-rule will be
`<!ELEMENT tag ANY >`.
The one-unambiguity is checked according to [2].

At the mentioned website you can also find a similar program for learning regular tree languages, also see [9].

References

1. D. Angluin. Inference of reversible languages. *J. of the ACM*, 29(3):741–765, 1982. 64, 66, 71
2. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998. 72
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd. edition, 2001. 70
4. H. Fernau. Identification of function distinguishable languages. In H. Arimura, S. Jain, and A. Sharma, editors, *Proceedings of the 11th International Conference Algorithmic Learning Theory ALT 2000*, volume 1968 of *LNCS/LNAI*, pages 116–130. Springer, 2000. 64, 66, 70, 71
5. H. Fernau. k-gram extensions of terminal distinguishable languages. In *International Conference on Pattern Recognition (ICPR 2000)*, volume 2, pages 125–128. IEEE/IAPR, IEEE Press, 2000. 66
6. H. Fernau. Approximative learning of regular languages. In L. Pacholski and P. Ružička, editors, *SOFSEM'01; Theory and Practice of Informatics*, volume 2234 of *LNCS*, pages 223–232. Springer, 2001. 65
7. H. Fernau. Learning XML grammars. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition MLDM'01*, volume 2123 of *LNCS/LNAI*, pages 73–87. Springer, 2001. 65, 72
8. H. Fernau. Parallel communicating grammar systems with terminal transmission. *Acta Informatica*, 37:511–540, 2001. 65
9. H. Fernau. Learning tree languages from text. In J. Kivinen, editor, *Conference on Learning Theory COLT'02*, to appear in the *LNCS/LNAI* series of Springer. 71, 72
10. E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967. 64
11. J. Gregor. Data-driven inductive inference of finite-state automata. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(1):305–322, 1994. 64

12. C. de la Higuera. Current trends in grammatical inference. In F. J. Ferri et al., editors, *Advances in Pattern Recognition, Joint IAPR International Workshops SSPR+SPR'2000*, volume 1876 of *LNCS*, pages 28–31. Springer, 2000. 64
13. V. Radhakrishnan. *Grammatical Inference from Positive Data: An Effective Integrated Approach*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay (India), 1987. 64, 66, 68
14. V. Radhakrishnan and G. Nagaraja. Inference of regular grammars via skeletons. *IEEE Transactions on Systems, Man and Cybernetics*, 17(6):982–992, 1987. 64, 65
15. R. E. Tarjan and J. van Leeuwen. Worst-case analysis of set union algorithms. *J. of the ACM*, 31:245–281, 1984. 70