CIS-TR-80-7

The Use of Tree Derivatives and a Sample Support Parameter for Inferring Tree Systems

by

Barry Levine

Computer and Information Science Department

University of Oregon

Eugene, Oregon 97403

THE USE OF TREE DERIVATIVES AND A SAMPLE SUPPORT PARAMETER FOR INFERRING TREE SYSTEMS

by

Barry Levine

Abstract - Tree systems have been studied in the theoretical setting as an extension of finite automata. They have been found useful in the practical domain when applied to syntactic pattern recognition. The practical applications of tree systems have motivated the examination of inference techniques for tree grammars and tree automata. In this paper we present a tree automaton inference algorithm which incorporates three concepts - tree derivatives, grammatical expansion and inferential strength. Tree derivatives are used for comparing tree forms. Grammatical expansion is a feedback mechanism which effectively enlarges the user sample. An inferential strength parameter is input by the user to indicate the amount of support required from the sample for inferences. The algorithm is also applied for inferring finite-state machines. Finally, we address an open problem posed by Joshi and Levy by demonstrating the use of our algorithm for the design of programming languages.

1. Introduction

Tree systems have been studied in the theoretical setting as an extension of finite automata, [7] and [11]. They have been found useful in the practical domain when applied to syntactic pattern recognition [2] and [18]. The practical applications of tree systems have motivated the examination of inference techniques for tree grammars and tree automata [8] and [12]. This paper deals with extending the results in Levine [17] by inferring tree automata using tree derivatives. The ideas of grammatical expansion and inferential strength are introduced. The inferential strength parameter specifies the support required from the sample to perform inferences. Grammatical expansion is the feedback technique wherein inferences determined at an earlier stage of the process are used to help form inferences at later stages.

These concepts are applied to yield an algorithm for inferring finite-state machines. We address the open grammatical inference problem for context-free languages, posed by Joshi and Levy [15], by applying our inference technique to the design of programming languages. It is shown that our method uses information more effectively (i.e. our method requires smaller samples) than the programming language inference algorithm by Crespi-Reghizzi, Lichten and Melkanoff [10].

In Section 2 we present preliminary definitions and results on tree systems. We also review the tree inference algorithms by Brayer and Fu [8], Edwards, Gonzalez and Thomason [12] and Levine [17]. The inference algorithm using tree derivatives, grammatical expansion and inferential strength is introduced in Section 3. Section 4 deals with space and time efficiency improvements to the inference technique. The algorithm is applied to finite-state inference in Section 5. Finally,

in Section 6 we address Joshi and Levy's open problem by demonstrating the use of skeletal structure samples with our algorithm for the design of programming languages. Our method is compared with the method by Crespi-Reghizzi, Lichten and Melkanoff in the same section.

2. Preliminaries

In this section we give the standard definitions for tree systems [7]. Then we present the theory upon which we based our inference technique. Finally, we conclude this section with a brief review of current results on the inference of tree systems.

Definition 2.1: Let N be the set of nonnegative integers and V the free monoid generated by N with 0 the identity and "." the operation. We define $a \leq b$ for $a,b \in V$ iff there exists $x \in V$ such that $a \cdot x = b$. a and b are *incomparable* iff $a \not\leq b$ and $b \not\leq a$. $a < b$ iff $a \leq b$ and $a \neq b$.

Definition 2.2: D is a *tree domain* iff it satisfies

i) $D \subseteq V$ and D is finite

ii) $b \in D$ and $a < b$ implies $a \in D$

iii) $a \cdot m \in D$ implies $a \cdot n \in D$ for $1 \leq n \leq m$, $n \in N$.

A *direct successor (direct predecessor)* of a node x is a node y where $y = x \cdot a$ ($y \cdot a = x$) for $a \in N$. A *terminal node* in D is one that has no direct successors.

Intuitively, a tree domain, D, is a finite, rooted, ordered tree such that a node in D implies all of its predecessors are in D and if an m^{th} descendant, x, is included in D then the m-1 siblings of x must also be in D. The elements of a domain allow one to uniquely address any node in the tree associated with the domain.

Definition 2.3: The *depth* of $a \in V$ is defined recursively as

depth(a)=0 if a=0

depth(a·x)=depth(a)+1 for $x \in N$

(i.e., if $a \neq 0$ then depth(a) is 1 plus the number of .'s). If t is a tree domain then depth(t)=max{depth(i): $i \in t$}.

The next definition describes labelled trees which associate information with the nodes of a tree domain.

Definition 2.4: A *tree* over the finite alphabet A ($t \in A^T$) is a mapping $t: D \rightarrow A$ which labels the nodes of the tree domain D. $t(x)=\varepsilon$ if $x \notin D$. The set of all terminal nodes of the labelled tree, t, concatenated left to right is the *frontier* of t, denoted fr(t). Given a labelled tree it is frequently required to reference or replace subtrees.

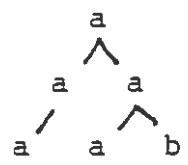
Definition 2.5: If $a,b,b' \in V$ and b=a·b' then b/a=b'. If $t \in A^T$ then the *subtree* of t at a where a is in the domain of t ($a \in Dom(t)$) is defined as $t/a=\{(x,b): (a \cdot x,b) \in t\}$. The *replacement of the subtree* at a with the subtree u, denoted $t(a \leftarrow u)$, is defined as $t(a \leftarrow u)=\{(b,x): t(b)=x$ and $a \not< b\} \cup \{a \cdot y: y \in Dom(u)\}$.

Example 2.1

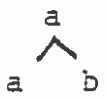
D = {0,1,1.1,2,2.1,2.2} U ={0,1,2}

A = {a,b}

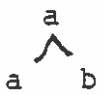
t:



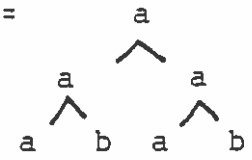
u:



t/2 =



t(1←u) =



Depth(t)=2, fr(t)=aab

Definition 2.6: A finite *tree automaton*, M, is an automaton which, given the states of all direct successors of a node, n, and the label of n, assigns a state to n. If the state of each node is uniquely determined then M is *deterministic*; otherwise M is *nondeterministic*. M is formally defined as a 4-tuple by M=(Q,A,f,F) where

Q is a finite set of states,

A is the finite set of tree labels,

F⊆Q is the set of final or accepting states and

f: Q^*xA→Q is the state assignment function.

$f=\{f_a: a\in A \text{ and } f_a: Q^*\rightarrow Q\}$.

Since M reduces (assigns states to) frontier nodes first and the root node last it is also known as a *frontier-to-root automaton* . If the state assigned to the root node of a tree t is in F then M accepts t and t∈T(M), the set of trees accepted by M. Henceforth, we will only consider deterministic tree automata since every nondeterministic tree automaton can be converted to an equivalent deterministic tree automaton [7].

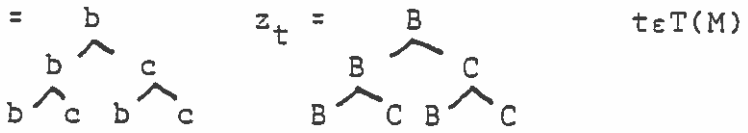
If M operates on t and z_t is a tree showing the reductions of all the nodes of t then z_t is the *state tree* for t.

Example 2.2

M=(Q,A,f,F) where

Q={B,C}, A={b,c}, f_b=B, f_c=C

f_b(B,C)=B, f_c(B,C)=C, F={B}



M accepts all rooted binary trees with left successor nodes labelled b, right successor nodes labelled c and root nodes labelled b.

We extend f, which operates on trees in A^T of depth 0 or 1, to f* as follows:

$f^*(a) = f_a$ for aεA

$f^*(a(t_1 \cdots t_n)) = f_a(f^*(t_1),\ldots,f^*(t_n))$.

Note that $f^*(t) = z_t(0)$ for $t \varepsilon A^T$. Hereafter we will use the extension of f when referencing tree automata.

Whereas tree automata accept (parse) sets of trees, regular tree grammars generate tree languages.

Definition 2.7: A *regular tree grammar* over A^T is a 4-tuple

G=(V,A,P,S) where

V is a finite set of nonterminals,

P is a finite set of productions of the form $t \to u$ for $\{t,u\} \subseteq (V \cup A)^T$ and

S is a finite set of trees in $(V \cup A)^T$.

Tree t derives u by G, denoted $t \overset{}{\underset{G}{\Rightarrow}} u$, iff there exists $v \to w \in P$ and $a \in Dom(t)$ such that $t/a=v$ and $t(a \leftarrow w)=u$. G is not given when it is clear from context. $t \Rightarrow^* u$ iff $t=t_0 \Rightarrow t_1 \Rightarrow \ldots \Rightarrow t_n=u$ for $n \geq 0$. The language generated by G, denoted T(G), is defined $T(G)=\{u: u \in A^T, t \Rightarrow^* u \text{ and } t \in S\}$. If P satisfies $P=\{t \to u: t \in V, u=x \in A \text{ or } u=\overset{x}{\underset{x_1 \ldots x_n}{\wedge}} \text{ for } x \in A \text{ and } x_1, \ldots, x_n \in V\}$ then G is an *expansive* grammar.

Theorem 2.1 (Brainerd [7]): For each regular system, one can effectively construct an equivalent expansive system.

Henceforth, unless noted otherwise, we will only consider expansive grammars.

Example 2.3

The following expansive grammar generates the set of trees accepted by the tree automaton in Example 2.2.

G=(V,A,P,S)

V={B,C} , A={b,c}, S={B}

$B \to b \mid \overset{b}{\underset{B \quad C}{\wedge}}$

$C \to c \mid \overset{c}{\underset{B \quad C}{\wedge}}$

Theorem 2.2 (Doner [11]): Let M be a tree automaton and G an expansive tree grammar. One can effectively construct an expansive grammar G^M where $T(M)=T(G^M)$ and a tree automaton M^G where $T(G)=T(M^G)$.

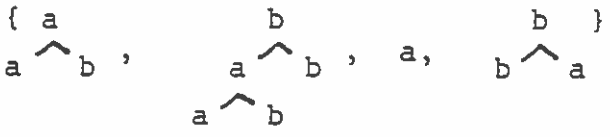
The concept of string derivatives [9] has been extended to trees [17].

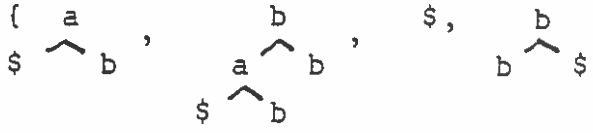
Definition 2.8: The m^{th} *order tree derivative* of the set of trees S with respect to the tree t, denoted $D_t^m(S)$, is recursively defined by $D_t^1(S) = \{u(b \leftarrow \$): u \in S \text{ and } u/b = t\}$ and

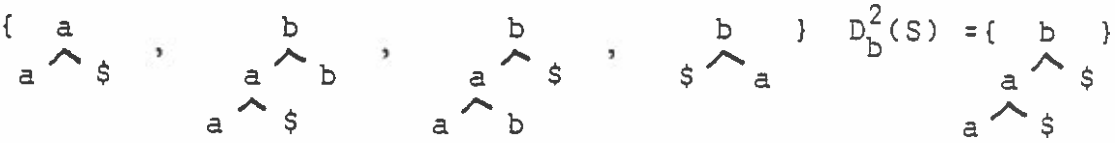
$D_t^{i+1}(S) = D_t^1(D_t^i(S))$ for $i \geq 1$, $S \subseteq A^T$ and $\$ \notin A$.

Hence, the m^{th} order tree derivative is the replacement of exactly m occurrences of t in each tree in S with a \$.

Example 2.4

$S = \{$  $\}$

$D_a^1(S) = \{$  $\}$, $D_a^2(S) = \emptyset$

$D_b^1(S) = \{$  $\}$ $D_b^2(S) = \{$ $\}$

$D_a^1(S) = \{\$,$  $\}$

The following results - Lemma 2.3, Theorem 2.4, Corollary 2.5, and Lemma 2.6 - are used as a basis for the inference algorithm given in Section 3. They are drawn from Levine [17].

Lemma 2.3: If $D_t^1(S) = D_u^1(S)$ then $D_t^j(S) = D_u^j(S)$ for $j \geq 1$.

Subtree-invariant equivalence relations, which are closely associated with tree derivatives, are defined as follows.

Definition 2.9: R is a *subtree-invariant equivalence relation* on trees in A^T iff tRu implies $v(x \leftarrow t)Rv(x \leftarrow u)$ for each $x \in Dom(v)$ and $t,u,v \in A^T$. A *state-minimizing* subtree-invariant equivalence relation of finite index for the tree automaton M was determined in [17]: tR^Mu iff for each $v \in A^T$ and each $x \in Dom(v)$, $v(x \leftarrow t) \in T(M)$ exactly when $v(x \leftarrow u) \in T(M)$.

Theorem 2.4: $D_t^1(T(M)) = D_u^1(T(M))$ iff tR^Mu.

Corollary 2.5: $D_t^j(T(M)) = D_u^j(T(M))$ iff tR^Mu where $j \geq 1$.

Our method of inference is based on examining depth bounded tree derivative sets.

Definition 2.10: The k *depth bounded tree derivative of order n* with respect to t is defined as

$$D_t^{k,n}(S) = \{u: u \in D_t^n(S) \text{ and } depth(u) \leq k\}.$$

Lemma 2.6 follows directly from Corollary 2.5.

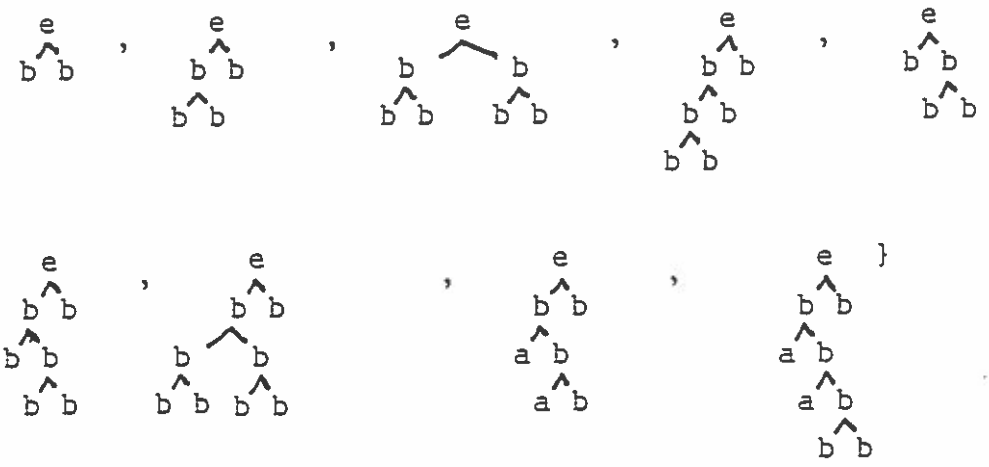
Lemma 2.6: tR^Mu iff $D_t^{k,i}(T(M)) = D_u^{k,i}(T(M))$ for $k \geq 0$ and $i \geq 1$.

We conclude this section with a brief review of the current results on the inference of tree automata. The algorithms input finite positive tree samples and then infer automata which accept supersets of the samples.

Definition 2.11: A *finite positive tree sample* S+ for a tree automaton M is a finite subset of T(M).

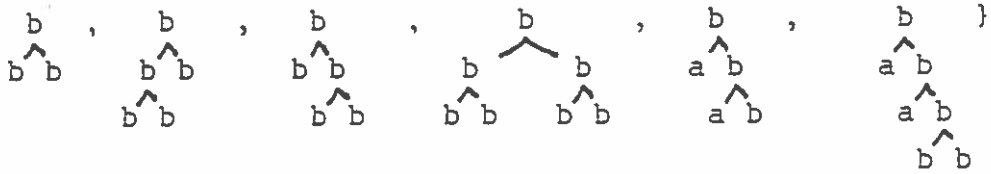
Brayer and Fu's algorithm [8] inputs S+ and a depth parameter k. The algorithm examines trees top down and forms tree sublanguages based on the set of successor subtrees to various nodes and positioning information. Sublanguages are considered to be equivalent (i.e., the same nonterminal is assigned to generate them) if they agree on all trees of depth less than or equal to k.

Example 2.5 (Brayer and Fu [8])

S+ = {  }

k=2

The left successor sublanguages of e are formed from

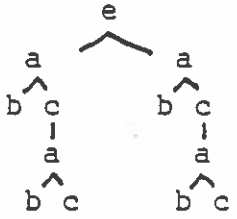
T = { b,  }

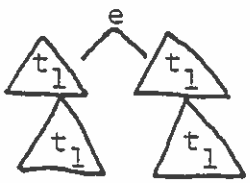
Since k=2 we only consider B={b,  }. The right successor sublanguages of e are also formed from B. Therefore, we obtain the rules S→  , B→b , B→  . We select the sublanguages A={a} and

$C = \{$  $\}$ and note that $T^2 \equiv B$ to obtain:

The inference technique by Edwards, Gonzalez and Thomason [12] uses the properties of self-embedding and regularity to produce a concise grammatical description of S+. Their algorithm locates repetitive sub-structures in S+ for determining recursive grammatical rules.

Example 2.6: (Edwards, Gonzalez and Thomason [12])

$t =$  $S+ = \{t\}$

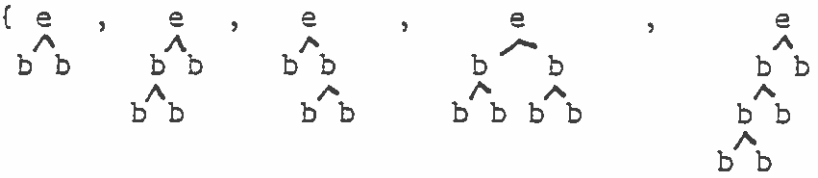
t is rewritten as:  $t_1 =$ 

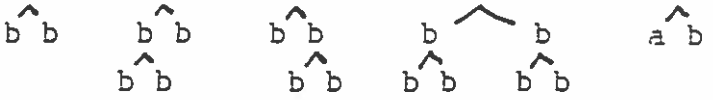
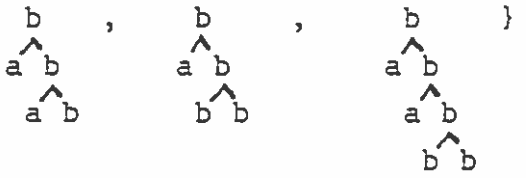
The grammar produced for S+ is:

The inference algorithm in Levine [17], denoted I1, examines the derivative sets $D_t^{k,i}(S+)$ with respect to the subtrees (t) of trees in S+ and infers the relation $tR^M u$ whenever $D_t^{k,i}(S+) = D_u^{k,i}(S+)$, $D_t^{k,i}(S+) \neq \emptyset$ and $D_t^{k,i}(S+) \neq \$$.

Example 2.7

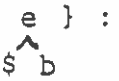
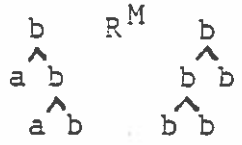
(S+ is the same sample as the sample given in Example 2.5)

S+= {  }

R^M=S+ U { b,  , a,  }

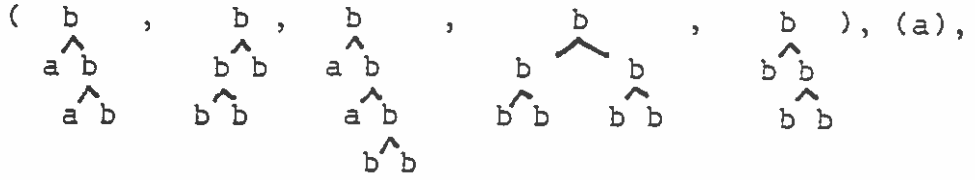
Il determines the following set of relations:

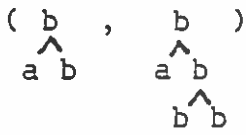
$D_b^{1,1}(S+)=D^{1,1}(S+)$ = {  } : bR^M 

$D_b^{1,1}$ (S+) = $D_b^{1,1}$ (S+) = {  } : 

...

Finally, R^M={(b, ) ,





M = ({S,A,B,C} , {e,a,b} , f, {S})

f_b = B, f_a = A, f_b(A,B)=C, f_b,(A,C)=B,

f_b(B,B)=B, f_e(B,B)=S

M is the same machine inferred by Brayer and Fu's algorithm with a depth parameter of 2. Note that I1 infers relations when the sample exhibits trees similar in form.

In Levine [17] we found I1 to be less sensitive than Brayer and Fu's technique to depth variations of sample trees. The limitation (not present in I1) of the method by Edwards, et al was the regularity requirement for samples presented to their systems.

3. A Second Tree Derivative Inference Algorithm

In this section we will modify I1 to incorporate two new concepts - grammatical expansion and inferential strength. Inference by grammatical expansion uses feedback to effectively enlarge the sample originally input. The use of inferential strength allows inferences to be made (two subtree-invariant equivalence classes merged) only when the derivative sets involved in the inference overlap according to a priori conditions set by the user.

The inference schemes reviewed in Section 2 required the user to present a finite positive sample S+. Then the methods attempted to generalize on S+ using the structural information contained therein. The grammatical expansion method is performed in stages. During each stage an inference is made on the original sample - S+ - in addition to its extension obtained via inferences from previous stages. The method uses feedback until the expanded S+ can no longer change with any further inferences.

We will clarify this notion by considering the modifications to I1 to incorporate grammatical expansion. In I1 the target automaton is determined from the subtree-invariant equivalence relation R^M. R^M was inferred from the derivatives $D^{k,n}_{t_j}(S+)$ for subtrees t_j in S+. Henceforth, we will abbreviate R^M by R since R^M is the only subtree-invariant equivalence relation that is considered. Note that R is initially the identity equivalence relation - all classes contain only one element. When $D^{k,n}_{t_i}(S+) = D^{k,n}_{t_j}(S+)$ the classes containing t_i and t_j are merged. The interpretation of this merger is the following. If $t_i(t_j)$ appears as a subtree at 'a' of some tree, t, accepted by the target automaton M (i.e., $t/a = t_i$, $t \varepsilon T(M)$) then $t(a \leftarrow t_j) \varepsilon T(M)$. The generalization on S+ by R (i.e., the inference of tRu) is denoted R(S+).

Our original concern (I1) was in finding elements of $D^{k,m}_t(S+)$ that were also in $D^{k,m}_u(S+)$. In the modified algorithm we will attempt to determine if elements of $D^{k,m}_t(S+)$ with \$-nodes replaced by u are in R(S+) or elements of $D^{k,m}_u(S+)$ with \$-nodes replaced by t are in R(S+). Thus, we will attempt to determine if the forms yielded by the original derivatives are the same

when R is applied.

This problem may be couched in terms of tree generating systems in the following manner.

Definition 3.1: The *regular tree grammar* $G_R=(V,A,P_R,S)$ *over* A^T *based on* R is defined as

V: $\emptyset$

A: the labels in trees found in S+

P_R: $\{(t,u): (t,u)\varepsilon R\}$

S: S+

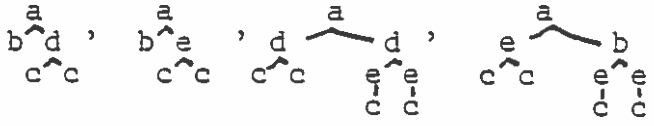
The algorithm must check whether given trees are generated by G_R. It is clear that an effective procedure exists for determining "$t\varepsilon T(G_R)$?". It is critical to note that derivatives are not taken over the complete expanded sample R(S+), but are taken over S+. Since trees in S+ were originally presented by the user, we restrict our interest in the derivative sets to trees of forms similar to $D_t^{k,m}(S+)$ rather than $D_t^{k,m}(R(S+))$.

Definition 3.2: Let
$SUB(D_t^{k,i}(S),u,R) = \{v: v\varepsilon D_t^{k,i}(S),\ v/a_1 = \ldots = v/a_i = \$ \text{ and } v(a_1 \leftarrow u)\ldots(a_i \leftarrow u)\varepsilon R(S)\}$. Then the *u-substitution set* for $D_t^{k,i}(S)$ with respect to R, denoted $N(D_t^{k,i}(S),u,R)$, is defined as

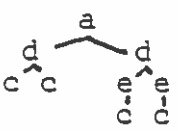
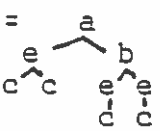
$$N(D_t^{k,i}(S),u,R) = \begin{cases} SUB(D_t^{k,i}(S),u,R) \text{ if } SUB(D_t^{k,i}(S),u,R) \neq \$ \\ \emptyset \text{ otherwise} \end{cases}$$

The revised inference algorithm uses substitution sets as well as derivative sets to make inferences--(t,u) is added to R (tRu) iff $N(D_t^{k,i}(S+),u,R) = D_t^{k,i}(S+) \neq \emptyset$, $N(D_u^{k,i}(S+),t,R) = D_u^{k,i}(S+) \neq \emptyset$.

Example 3.1

$S+ = \{$  $\}$

$t_1 = b$, $t_2 = c$, $t_3 =$  , $t_4 =$  , $t_5 =$, $t_6 =$  ,

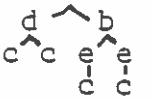
$t_7 =$  , $t_8 =$  , $t_9 =$  , $t_{10} =$  , $t_{11} =$ 

$R = \{(t_i, t_i): i = 1,\ldots,11\}$

$D^{1,1}_{t_3}(S+) = D^{1,1}_{t_4}(S+) = \{$  $\} = N(D^{1,1}_{t_3}(S+), t_4, R) = N(D^{1,1}_{t_4}(S+), t_3, R)$

$R = R \cup \{(t_3, t_4)\}$

$D^{2,1}_{t_6}(S+) = \{$  $\}$

Substituting t_7 for \$ in  we obtain  $\varepsilon R(S+)$

since $t_3 R t_4$. Therefore, $D^{2,1}_{t_6}(S+) = N(D^{2,1}_{t_6}, t_7, R)$. Likewise,

$D^{2,1}_{t_7}(S+) = \{$  $\} = N(D^{2,1}_{t_7}(S+), t_6, R)$ so $R = R \cup \{(t_6, t_7)\}$.

There are no further inferences possible using the revised algorithm so the relations $t_3 R t_4$ and $t_6 R t_7$ are output. Note that I1 would determine only $t_3 R t_4$.

The requirement that the substitution sets completely match the derivative sets might be too restrictive. In many cases there may be structural information present in the sample, but not very well represented therein. We will add a feature that performs inferences when $N(D_t^{k,i}(S+),u,R)$ and $D_t^{k,i}(S+)$ overlap by a given amount and similarly for $N(D_u^{k,i}(S+),t,R)$ and $D_u^{k,i}(S+)$.

Definition 3.3: The *strength of inductive information relating t and* u, when R and S+ are given, denoted Stren (t,u,R,S+), is defined as

$$\text{Stren}(t,u,R,S+) = \max_{k,i} \left[\frac{|N(D_t^{k,i}(S+),u,R)\ |+|\ N(D_u^{k,i}(S+),t,R)|}{|D_t^{k,i}(S+)\ |+|\ D_u^{k,i}(S+)|} \right]$$

Note that the inference algorithm using grammatical expansion required Stren(t,u,R,S+) = 1 for all inferences. We will relax this requirement by introducing a strength parameter into the algorithm which will allow strengths less than 1. The new method will perform inferences whenever the calculated strengths (percentage of overlap between the substitution sets and derivative sets) are at least as large as the strength parameter. The formal specification of the inference algorithm using grammatical expansion and inferential strength follows.

ALGORITHM I2: Inference of Tree Automata using Tree Derivatives, Grammatical Expansion and a Strength Parameter

1. S+, a finite positive tree sample for the target tree automaton M, is input.
2. Strn, the strength parameter is input.
3. For all tree pairs (t,u) not in the same class of R^M Do
   If Stren(t,u,R,S+) $\geq$ Strn
   Then Merge the classes containing t and u.

4. $R^{M'}$, the subtree-invariant equivalence relation obtained from the application of Step 3 to R^M, is output.

Algorithm I2 outputs the inferred automaton, M', as $R^{M'}$ since M' is easily determined from $R^{M'}$.

Example 3.2

$S+ = \{ \overset{a}{b\hat{}d}, \overset{a}{c\hat{}g}, \overset{a}{c\hat{}e}, \overset{a}{b\hat{}g} \}$

$D_b^{1,1}(S+) = \{ \overset{a}{\$\hat{}d}, \overset{a}{\$\hat{}g} \}$, $D_c^{1,1}(S+) = \{ \overset{a}{\$\hat{}g}, \overset{a}{\$\hat{}e} \}$

$D_d^{1,1}(S+) = \{ \overset{a}{b\hat{}\$} \}$, $D_e^{1,1}(S+) = \{ \overset{a}{c\hat{}\$} \}$, $D_g^{1,1}(S+) = \{ \overset{a}{c\hat{}\$}, \overset{a}{b\hat{}\$} \}$.

Suppose Strn = 2/3 and R^0 is the purely reflexive relation. The non-zero strengths in order of computation are:

Stren(b,c,R^0,S+) = 2/4,

Stren(d,g,R^0,S+) = 2/3 → $R^1 = R^0 \cup \{(d,g)\}$

Stren(e,g,R^1,S+) = 2/3 → $R^2 = R^1 \cup \{(e,g)\} = \{(b),(c),(d,e,g)\}$

Stren(b,c,R^2,S+) = 1 → $R^3 = R^2 \cup \{(b,c)\} = \{(b,c), (d,e,g)\}$

R^3 is output by I2 since no further inferences are possible. The tree automaton determined by R^3 is

M = ({A,B,D}, {a,b,c,d,e,g} , f, {A}) where

$f_b = f_c = B$, $f_d = f_e = f_g = D$, $f_a(B,D) = A$

Note that I1 would output R^0. If Strn was set to any value less than 2/3 then I2 would still output R^3. Brayer and Fu's algorithm could not make any inferences.

4. Efficiency Improvements to Algorithm I2

In this section we will discuss modifications to the relation set and the method for checking whether trees are in $T(G_R)$. These changes will improve the space required (storage for R is decreased) and the time necessary for completion of an inference (the times for checking trees in $T(G_R)$ and derivative set comparisons are decreased).

The relation set is changed as follows: If $\bar{t} = \{t_1, \ldots, t_n\}$ is an explicitly specified equivalence class then choose a member of that class with minimum depth as its representative. Do the same for all classes. If $t_i \in \bar{t}$ and u_j is a subtree of t_i where $u_j \in \bar{u}$ then substitute u for all occurences of u_j in t_i. Repeat this process until no further substitutions are possible. If this process results in a merging of two classes then the new class representative is chosen as before and substitutions are made again (e.g. if $R = \{ a, \overset{a}{a\hat{}c}\ ;\ c, \overset{c}{a\hat{}b}\ ,$ $\overset{a}{\underset{a\hat{}b}{a\hat{}c}}\ \}$ then $\overset{a}{\underset{a\hat{}b}{a\hat{}c}}$ is reduced to $\overset{a}{a\hat{}c}$ which is in $\bar{a}$ so $\bar{a}$ and $\bar{c}$ are merged to yield $R = \{ a, \overset{a}{a\hat{}c}\ , c, \overset{c}{a\hat{}b}\ \}$). Repeat the class merging and substitution process until no more changes are possible. This results in a relation set of minimum depth, an elimination of redundant information and a potential decrease in the number of iterations of STEP 3 in I2. Note that all subtrees of trees in R are class representatives. The inference algorithm I2 using M1 is denoted I2(M1).

The second change, denoted M2, involves the tree grammar $G_R = (V,A,P_R,S+)$. Recall $P_R = \{(t,u): (t,u)\varepsilon R\}$. Instead of S+ use S' which is derived from S+ using the reductions $t \rightarrow u$ for $(t,u)\varepsilon R$ and t an element of the class represented by u. Thus, if $(a, \overset{a}{a\hat{}b})\varepsilon R$ and $\overset{x}{\substack{a\hat{}a \\ a\hat{}b \\ a\hat{}b}} \varepsilon$ S+ then $\overset{x}{a\hat{}a} \varepsilon$ S'. Clearly, $T(G_R) = T(G'_R)$ where $G'_R = (V,A,P_R,S')$. It is also evident that the productions of G_R can be altered to $P' = \{(t,u)$: $(t,u)\varepsilon R$ and u is an element, other than t, of the class represented by t$\}$ without affecting the result of the inference process.

At present we have the basic inference algorithm I2 with modifications M1 and M2 applied to it. This algorithm will be denoted I2(M1,M2). M2 speeds up the checking process for $t\varepsilon T(G_R)$ since the productions in G_R are non-depth decreasing. This property in P' enables one to form an expansive tree grammar G" that generates the same set of trees as G_R and a tree automaton M_R where $T(G_R) = T(G") = T(M_R)$. Thus, the formation of M_R is done in a straightforward fashion. Furthermore, since P' is invertible (i.e. $t \rightarrow u\varepsilon P'$ and $v \rightarrow u\varepsilon P'$ implies $t=v$) then M_R is deterministic. Therefore, instead of checking "$t\varepsilon T(G_R)$?" we can deterministically parse t to answer "$t\varepsilon T(M_R)$?".

The use of M_R with M2 and M1 by I2, denoted I2(M1,M2,M_R), speeds up the checking process. An additional benefit of using M_R is at the

conclusion of I2(M1,M2,M_R) both the set of inferred relations - R - and the inferred automaton - M_R - are already formed.

In summary, the changes to the representation of R are the use of class representatives, the elimination of redundant information and, possibly, the merging of classes at an earlier time than I2. These changes in R save space and time (if classes are merged then derivative sets need not be checked). We have also changed the checking process from "tεT(G_R)?" to "tεT(M_R)?" where M_R is deterministic. This reduces the checking time and makes the currently inferred results - M_R - available at any time for examination or output.

5. Inference of Finite-State Machines

When S+ is restricted to linear trees - every node has exactly one successor or is a terminal - then the inferred tree automaton is equivalent to a finite-state machine. In this section we will compare our inference method with the inference algorithm for finite-state machines by Biermann and Feldman [3], denoted IBF. IBF inputs S+ and a length parameter k. Then it computes first order derivative sets with respect to the prefixes of strings in the sample. The derivative sets for the strings s and t are *k-equivalent* ($D_s^1(S+) \overset{k}{=} D_t^1(S+)$) if $D_s^1(S+)$ and $D_t^1(S+)$ agree on all strings of length less than or equal to k (disregarding the \$ marker). sR^Mt is inferred whenever $D_s^1(S+) \overset{k}{=} D_t^1(S+)$.

Example 5.1

$S+ = \{01,100,111,0010\}$ $k=2$

$D_0(S+) \overset{2}{\equiv} D_{11}(S+) \overset{2}{\equiv} \{1\}$

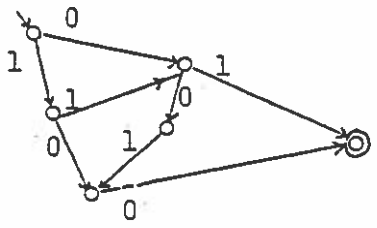
$D_1(S+) \overset{2}{\equiv} \{00,11\}$, $D_{10}(S+) \overset{2}{\equiv} D_{001}(S+) \overset{2}{\equiv} \{0\}$

$D_{00}(S+) \overset{2}{\equiv} \{10\}$

$D_{01}(S+) \overset{2}{\equiv} D_{100}(S+) \overset{2}{\equiv} D_{111}(S+) \overset{2}{\equiv} D_{0010}(S+) \overset{2}{\equiv} \{\lambda\}$

$R = \{(\lambda), (0,11), (1), (10,001), (00), (01, 100, 111, 0010)\}$

M:



A shortcoming of IBF is when the sample exhibits inadequate information to perform inferences based on the k-parameter. The next example demonstrates the situation.

Example 5.2

$S+ = \{ab,cb,ae,cd,ad,ccb\}$

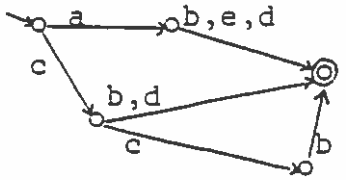
$D_a(S+) = \{b,e,d\}$, $D_c(S+) = \{b,d,cb\}$, $D_{cc}(S+) = \{b\}$

$D_{ab}(S+) = D_{ae}(S+) = D_{ad}(S+) = D_{cb}(S+) = D_{cd}(S+) = D_{ccb}(S+) = \{\lambda\}$

For any k setting we obtain:

ab R ae R ad R cb R cd R ccb

M:



Note that I2 applied to S+ with Stren = 4/5 would determine aRc. Thus, I2 could act on the overlap between the sets of derivatives with respect to a and c.

The next example shows the capability of I2 to extract information from the sample that is not readily apparent upon visual inspection (and therefore cannot be obtained by IBF).

Example 5.3

S+ = {acd,bcd,axef,bye,aefg,aeg,axcd}

$D^1_a(S+)$ = {cd,xef,efg,eg,xcd}, $D^1_{ac}(S+)$ ={d}

$D^1_{ax}(S+)$ = {ef,cd}, $D^1_{axe}(S+)$ ={f}, $D^1_{axc}(S+)$ ={d}

$D^1_{ae}(S+)$ ={fg,g}, $D^1_{aef}(S+)$ ={g}

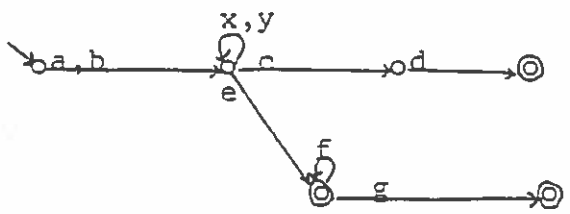
$D^1_b(S+)$ = {cd,ye}, $D^1_{bc}(S+)$ ={d}, $D^1_{by}(S+)$ ={e}

ac R axc R bc, ae R aef (1)

When Stren = 1/2 I2 determines

a R b, a R ax, ax R by

M:



Both the strength factor and grammatical expansion helped to combine classes of R to simplify M. IBF could only determine the relations (1) by using k.

6. The Design of Programming Languages

In this section we address the open question posed by Joshi and Levy[15] by demonstrating the use of skeletal descriptions for inferring context free grammars. In[10] Crespi-Reghizzi, Melkanoff and Lichten specified an inference system for the design of operator precedence programming languages. We will briefly review their technique and compare it to I2. We find that our method can use structural information in the samples more effectively, thereby requiring smaller samples for inferences.

Crespi-Reghizzi, et al specified an inference system that used a sample consisting of skeletal structural descriptions[15]. These structural descriptions (produced by a parenthesis grammar for the target language) are derivation trees for the sentences in the target language with nonterminal nodes not labelled. Their method is essentially a labelling procedure for these nodes.

Definition 6.1: The *parenthesized version,* [G], of G is formed from the productions A→[X] where A→X is in G. [and] are symbols not already in G.

Definition 6.2: The *skeleton*, s, of a tree, t, is the same as t with all internal nodes labelled with ".".

Their algorithm inputs derivation tree skeletons from [G] and uses terminal profiles to label internal nodes.

Definition 6.3: The *left* and *right terminal sets* of order k of a tree t are defined as:

$L_k(t)$ = {u: z is the frontier of t and u is the prefix of length k of z}

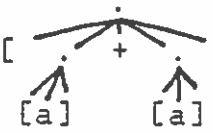
$R_k(t)$ = {u: z is the frontier of t and u is the suffix of length k of z}

If $|z|<k$ then u = z.

The *terminal profile* $P_k(t)$ *of order* k of a tree t is defined as

$P_k(t) = \lceil L_k(t);\ R_k(t) \rceil$

Example 6.1

t = 

z = [[a] + [a]]

$L_1(t)$ = {[} , $R_1(t)$ = {]}, $P_1(t) = \lceil \{[\}; \{]\} \rceil$

$L_5(t)$ = {[[a]+}, $R_5(t)$ = { +[a]] }

$P_5(t) = \lceil \{[[a]+\}\ ;\ \{+[a]]\} \rceil$

For clarity, we rewrite $P_5(t)$ as $\lceil [[a]+;\ +[a]] \rceil$

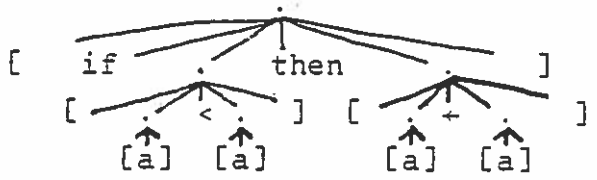
The algorithm by Crespi-Reghizzi, et al inputs a sample set of skeletons, $S+ = \{S_1, \ldots, S_n\}$, and a length parameter k. Then all the internal nodes of trees in S+ are labelled with P_k.

Example 6.2

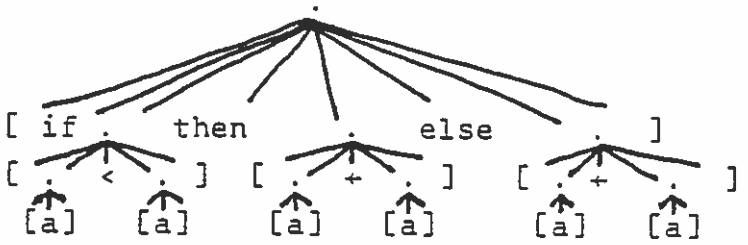
We apply their algorithm to infer a grammar for "if-then" and "if-then-else" statements. The following three sentences are supplied (with the usual association of statement parts):

1. If a<a then a←a
2. If a<a then a←a else a←a
3. If a<a then if a<a then a←a else a←a

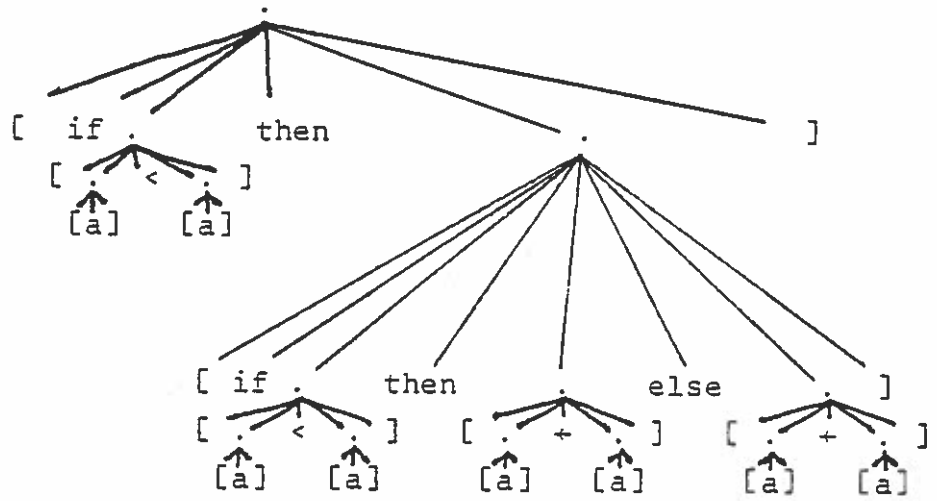
S_1 =



S_2 =



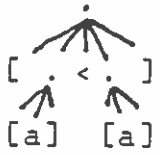
S_3 =

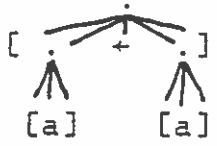


$S+ = \{S_1, S_2, S_3\}$

If k<5 then a<a and a←a reduce to the same variable. Let k = 5:

A: P_5() = ⌈[a]; [a]⌉ A→a

B: P_5 () = ⌈[[a]<; <[a]]⌉ B→A<A

C: P_5() = ⌈[[a]←;←[a]]⌉ C→A←A

D: $P_5(S_1) = P_5(S_2)$ = ⌈[if [[a; [a]]]⌉ D→if B then C

D→if B then C else C

E: $P_5(S_3)$ = ⌈[if [[a; a]]]⌉ E→if B then D

For $5 \leq k \leq 10$ there are no changes.

Let k = 11:

A, B and C are the same.

D: $P_{11}(S_1)$ = ⌈[if [[a] < [a]]; then [[a]←[a]]]⌉

D→if B then C

E: $P_{11}(S_2)$ = ⌈[if [[a]<[a]]; else [[a]←[a]]]⌉

E→if B then C else C

F: $P_{11}(S_3)$ = ⌈[if [[a]<[a]]; [[a]←[a]]]]⌉

F→if B then E

The complete grammar inferred for k=11 is G=({S,A,B,C,D,E,F},

{a,if,then,else,<,←}, S , P)

P: S→D | E | F

A→a

B→A<A

C→A←A

D→if B then C

E→if B then C else C

F→if B then E

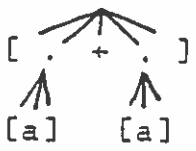
The inferred grammar does not change from G for values of k larger than 11.

The following example illustrates the application of I2 to the sample given in Example 6.2.

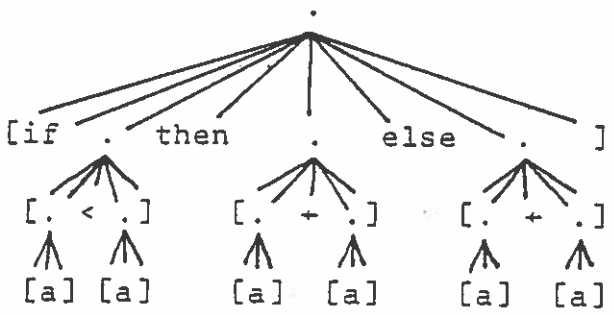
Example 6.3

S+ is the same as in Example 6.2. (The use of brackets is redundant when I2 is applied. However, we include them for clarity).

Let t =

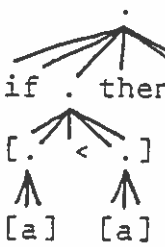


u =



, and Strength = 2/5.

Then $N(D_t^{3,1}(S+),u,R)$ = {[if . then $]} = $N(D_u^{3,1}(S+),t,R)$



and Stren(t,u,R,S+) = 2/5. tRu is the only generalization that can be made based on the structural content of S+. The tree automaton resulting from this inference is:

M = (Q,T,f,F) where

Q = {'[', 'a', ']',IF,THEN,ELSE,'<', '←',A,B,C,D}

T = {·, if, then, else, <, ←, [,], a}

F = {C,D}

$f_{[}$ = '[', f_{a}='a', $f_{]}$ = ']', f_{if} = IF, f_{then} = THEN

f_{else} = ELSE, $f_{<}$ = '<', $f_{\leftarrow}$ = '←'

f.('[', 'a', ']') = A

f.('[', A, '<', A, ']') = B

f.('[', A, '←', A, ']') = C

f.('[', IF, B, THEN, C, ELSE, C, ']') = C

f.('[', IF, B, THEN, C, ']') = D

The equivalent context-free grammar (non-parenthesized version) for M is G = ({A,B,C,D,S}, {a,<,←, if, then, else} , S, P)

S→C | D

P: A→a

B→A<A

C→A←A | if B then C else C

D→if B then C

Note that each "else" is grouped with the most recent "then".

Example 6.2 demonstrated that the amount of structural information used by their algorithm was dependent on the location of structures in strings. When k was set too low then structural information contained in the sample was not used (it appeared in the center of strings). As k was increased the algorithm split grammatical variables

(discriminated between strings that differed towards their centers rather than their ends). Thus, the manner of splitting did not allow the inference of recursive grammatical rules (higher values of k proved too discriminatory). This shortcoming was due to the fact that their algorithm viewed samples as structured strings ([10]) rather than skeletal structures for derivation trees.

In contrast to the properties of their algorithm mentioned above, we find that I2 uses all of the structural information present in trees involved in inferences. Thus, if tRu is inferred from $D_t^{k,i}(S+)$ and $D_u^{k,i}(S+)$ then the placement of all subtrees in $D_t^{k,i}(S+)$ and $D_u^{k,i}(S+)$ is relevant. Furthermore, by varying the strength parameter the user can observe all of the structural generalizations (variable mergers) supported by the sample.

7. Conclusion

In this paper we have proposed a method for automating the design of tree languages. It uses tree derivatives, feedback and interacts with the user via the strength parameter. The method was based on comparing tree forms, which helped determine the roles of subtrees. We have found that the method performs well when compared to the tree inference algorithms by Brayer and Fu and Edwards, Gonzalez and Thomason.

We have used the notion of generalizing finite automata to tree automata for applying our algorithm to the inference of finite automata. Finally, we have shown that by using skeletal structure samples - context-free derivation trees without variable labels - our algorithm can be used to aid in the design of programming languages.

An extension that is under study combines the method of locating repetitive substructures, given in [12], with our inference algorithm.

REFERENCES

[1] M.A. Arbib and Y. Give'on, "Algebra automata I: Parallel programming as a prolegomena to the categorical approach," Inform. Contr., vol. 12, pp. 331-345, April 1968.

[2] B.K. Bhargava and K.S. Fu, "Tree systems for syntactic pattern recognition," IEEE Trans. Comput., vol. C-22, pp. 1087-1099, Dec. 1973.

[3] A.W. Biermann and J.A. Feldman, "On the synthesis of finite-state machines from samples of their behavior," IEEE Trans. Comput., vol. C-21, pp. 592-597, June 1972.

[4] T.L. Booth and K.S. Fu, "Grammatical inference: Introduction and survey - Part I," IEEE Trans. Syst., Man, Cybern., vol. SMC-5, pp. 95-111, Jan. 1975.

[5] T.L. Booth and K.S. Fu, "Grammatical inference: Introduction and survey - Part II," IEEE Trans. Syst., Man, Cybern., vol. SMC-5, pp. 409-422, July 1975.

[6] W.S. Brainerd, "The minimalization of tree automata," Inform. Contr., vol. 13, pp. 484-491, Nov. 1968.

[7] W.S. Brainerd, "Tree generating regular systems," Inform. Contr., vol. 14, pp. 217-231, Feb. 1969.

[8] J.M. Brayer and K.S. Fu, "A note on the k-tail method of tree grammar inference," IEEE Trans. Syst., Man, Cybern., vol. SMC-7, pp. 293-300, April 1977.

[9] J.A. Brzozowski, "Derivatives of regular expressions," Journal Ass. Comput. Mach., vol. 11, pp. 481-494, Oct. 1964.

[10] S. Crespi-Reghizzi, M.A. Melkanoff and L. Lichten, "The Use of Grammatical Inference for Designing Programming Languages," CACM, vol. 16, no. 2, Feb. 1973.

[11] J. Doner, "Tree acceptors and some of their applications," J. Comput. Syst. Sci., vol. 4, pp. 406-451, Oct. 1970.

[12] J.J. Edwards, R.C. Gonzalez and M.G. Thomason, "An algorithm for the inference of tree grammars," Intern. Jour. Comput. Infor. Sci., vol. 5, no. 2, June 1976.

[13] T.G. Evans, "Grammatical inference techniques in pattern analysis," in Software Engineering, vol. 2, J.T. Tou, Ed. New York: Academic, 1971.

[14] J.E. Hopcroft and J.D. Ullman, Formal Languages and Their Relation to Automata. Reading, Mass.: Addison-Wesley, 1969.

[15] A.K. Joshi and L.S. Levy, "Skeletal structural descriptions," Inform. Contr., vol. 39, pp. 192-211, Nov. 1978.

[16] B.A. Levine, "The automated inference of tree systems," Ph.D. dissertation, Dept. Comput. Sci., Oregon State Univ., Corvallis, June 1979.

[17] B.A. Levine, "Derivatives of Tree Sets with Applications to Grammatical Inference," submitted for publication.

[18] K.L. Williams, "A Multidimensional Approach to Syntactic Pattern Recognition," Pattern Recognition, vol. 7, no. 3, pp. 125-137, Sept. 1975.

Index Terms

tree derivatives, inference of tree automata, grammatical expansion, inferential strength, inference of programming languages