Inference of high-dimensional grammars such as tree grammars and web grammars is discussed. The k-tall Inference procedure for finite-state grammars is extended to the case of regular tree grammars. The behavior of the k-tall procedure with variable values of k is studied. The derivation diagram of context-free web languages is introduced. A semantic teacher is used for the inference of web grammars. Application examples in picture and scene analysis are presented.
Inference of high-dimensional grammars such as tree grammars and web grammars is discussed. The k-tail inference procedure for finite-state grammars is extended to the case of regular tree grammars. The behavior of the k-tail procedure with variable values of k is studied. The derivation diagram of context-free web languages is introduced. A “semantic teacher” is used for the inference of web grammars. Application examples in picture and scene analysis are presented.

INTRODUCTION

The use of formal linguistics in modeling natural and programming languages and describing physical patterns and data structures has recently received increasing attention. Grammar or syntax rules are employed to describe the syntax of languages or the structural relations of patterns or data. In order to model a language or to describe a class of patterns or data structures under study more realistically, it is hoped that the grammar used can be directly inferred from a set of sample sentences or a set of sample patterns (or data). Grammatical inference is the problem of learning a grammar based on a set of sample sentences. Potential applications of grammatical inference include areas of pattern recognition, information retrieval, programming language design, translation and compiling, graphics languages, man-machine communication, and artificial intelligence.

In (1–3), inference of nonstochastic and stochastic string grammars was surveyed and a heuristic inference procedure for tree grammars was proposed in (4). In this paper, the k-tail presented. An inference procedure for transition network grammars was proposed in (4). In this paper, the k-tail inference procedure for finite-state grammar (5) is extended to the case of regular tree grammars. The behavior of the k-tail tree grammar inference method for varying values of k is studied. A web grammar interpretation of Winston’s structure learning is discussed and an inference procedure for context-free web grammars is suggested.

K-TAIL INFERENCE METHOD FOR REGULAR
TREE GRAMMARS

The k-tail inference method for finite-state string grammars requires an integer parameter k as input along with the presentation of (positive)
LEVEL

OFFICE OF SCIENTIFIC RESEARCH (AFSC)

NOTICE OF TRANSMITTAL TO DDC

This technical report has been reviewed and is approved for public release IAW AFR 190-12 (76). Distribution is unlimited.

A. D. BLOSE
Technical Information Officer
training samples (5). Sublanguages $S_w$ are created where

$$S_w = \{ x'w x \text{ is a string in the (positive) training set and}$$

$$| x' | \leq k \}$$

where $| x' |$ is the length of $x'$. Equivalent $S_w$ sets are then combined to form the $i$th sublanguage. A rule, $A_i \rightarrow t A_j$ is produced if there is a string $w$ such that $S_w$ is the $i$th sublanguage and $S_{w_t}$ is the $j$th sublanguage. The rule $A_i \rightarrow t$ is produced if there is a string $w$ such that $S_w$ is the $i$th sublanguage and $w_t$ is in the training sample. For strings, the exactness of the grammar produced for any given training set can be adjusted by varying $k$ from 0 up to the length of the longest string in the training set. The inferred languages vary correspondingly from something close to the universal language to the presentation itself. Thus, any method restricted to $k = 1$ will infer grammars which generate languages which are very “loose” in their fit of the sample set.

It is possible to extend the k-tail method for finite-state string grammars to regular tree grammars. The method is as follows:

**Step 1.** Form the following collection:

$$C_t = \{ ( \tau_1, \tau_2, \ldots, \tau_m ) | \tau_1 \tau_2 \ldots \tau_m$$

is a tree in the training set

$$\text{and } | \tau_0 | \leq k \text{ for } \ell = 1, 2, \ldots, m \}$$

where

- $t$ is a tree with a single special frontier node.
- $\tau_1, \tau_2, \ldots, \tau_m$ are any trees that can occur in positions 1, 2, ..., $m$.
- $\tau_1 \tau_2 \ldots \tau_m$ is the tree formed by concatenating $\tau_0$ at the $p$th position of the special frontier node of $t$.
- $| \tau_0 |$ is the depth of $\tau_0 + 1$
- $m$ is the number of descendants of $t$ and is not fixed to any particular integer.

Note that $t$, the empty tree, is possibly a member of $C_t$.

**Step 2.** The collection $C_t$ of tuples of trees can be partitioned into subcollections of $m$-tuples where $m$ is a fixed integer for all elements of each subcollection.

$$C_t = C_{10} \cup C_{11} \cup \ldots \cup C_{1m}$$

where

- $C_{10} = \{ \epsilon \}$ if $t$ is in the training set, otherwise $C_{10} = \emptyset$
- $C_{11} = \{ ( \tau_1 ) | \tau_1$ is a tree in the training set and $| \tau_1 | \leq k \}$
- $C_{12} = \{ ( \tau_1, \tau_2 ) | \tau_1 \tau_2$ is a tree in the training set and $| \tau_1 | \leq k$ and $| \tau_2 | \leq k \}$ (Note here that the subscript indicates the position a tree occupies and that $\tau_1$ in $C_{12}$ is not necessarily always the same tree nor is it the same tree as $\tau_1$ in $C_{11}$)
- $C_{1m} = \{ ( \tau_1, \tau_2, \ldots, \tau_m ) | \tau_1 \tau_2 \ldots \tau_m$ is a tree in the training set and $| \tau_0 | \leq k \text{ for } \ell = 1, 2, \ldots, m \}$

Thus, $C_t$ is a collection of tuples of trees and $C_{1i}$ is a collection of $i$-tuples of trees where $i$ is a fixed, specified integer.

Each of these collections defines all of the $i$-tuples of $k$-tail trees that are in the training set with root attached to the tree, $t$, at its special frontier node. The collections are separated in this way because an $i$-tuple and a $j$-tuple where $i \neq j$ cannot be generated by the same rule. Thus, we will now demonstrate the procedure that should be applied to each of the subcollections.

**Step 3.**

The next step is one which is not necessary in the case of strings. It is necessary here because a node can have several descendants and it may be that only certain ordered combinations of descendants are allowed. Thus, each subcollection of $i$-tuples of trees, $C_{1i}$, must be further divided into subcollections of $i$-tuples, each of which can be expressed as the cartesian product of $i$ sets of trees. Thus, $C_{1i}$ may be written:

$$C_{1i} = C_{1i1} \cup C_{1i2} \cup \ldots \cup C_{1im}$$

where

- $C_{1ij} = \{ ( \tau_1, \tau_2, \ldots, \tau_j ) \} \text{ if } \tau_i \in S_{ij}, \tau_j \in S_{jj}, \ldots \}$
- or $C_{1ij} = \{ ( \tau_1, \tau_2, \ldots, \tau_j ) \} \text{ if } \tau_i \in S_{ij} \times S_j \times \ldots \times S_{jj} \}$

That is, each $C_{1ij}$ is characterized by $i$ sets, $S_{ij}$, ($\ell = 1, 2, \ldots, i$), of trees from which the $\ell$th member of an $i$-tuple must be selected. These $S_{ij}$ sets are sublanguages of trees and may be regarded as a set of trees generated by a particular nonterminal of the tree grammar. The difficult part of this step is to find those sets $S_{ij}$ which efficiently characterize the $C_{1ij}$. First of all, the resulting grouping is not unique. One possible grouping would be that in which each $C_{1ij}$ has one element. This would not be a good choice because each $C_{1ij}$ will result in a grammar rule. Thus, this choice would result in a large number of rules. Since there are a finite number of elements in $C_{1ij}$, there are a finite number of groupings and each of these can be tried. It is not necessary that the $C_{1ij}$ be disjointed. A particular grouping would be optimum if it introduced a minimum number of new $S_{ij}$ sublanguages.

Now the rules for the grammar can be constructed. Equivalent $S_{ij}$ sublanguages are combined and a nonterminal is assigned corresponding to each distinct sublanguage. Now a rule $A_{1ij} \rightarrow x A_{1j1} A_{1j2} \ldots A_{1jm}$ is produced if there is a tree $t$ such that:
1. Aj is the nonterminal corresponding to the sublanguage Sj.
2. There exists a C_{tx} that contains the sublanguage Sj in the tth position of its specification.
3. tx is a tree with x concatenated at the tth position of t.
4. There exists a C_{txmn} which is specified by the sublanguages S_{t1}, S_{t2}, ..., S_{tm}.
5. A_1, A_2, ..., A_m are the nonterminals corresponding to the S_{tn1}, S_{tn2}, ..., S_{tmm} sublanguages, respectively.
6. Either x is a tree in Sj where \alpha \in S_{tn1}, \beta \in S_{tn2}, ..., \lambda \in S_{tmm} or |\alpha \beta \lambda| > k.

A rule Aj \rightarrow x is produced if conditions 1, 2 and 3 above are satisfied and tx is in the training set.
To illustrate consider the following example:

Example 1:
Consider the following regular tree grammar:

(1) \quad S \rightarrow S
(2) \quad B \rightarrow b
(3) \quad B \rightarrow b
(4) \quad A \rightarrow a
(5) \quad B \rightarrow b

The training set is the following:

(1) \quad S
(2) \quad S
(3) \quad S
(4) \quad S
(5) \quad S
(6) \quad S

Step 1:
(Note: Greek letters are used here to specify the distinct trees which were all represented by t in the explanation above.)

Now assume k = 1 and construct the grammar as follows:

Step 2:
\begin{align*}
C_\alpha &= C_{\alpha 0} = \phi \\
C_\beta &= C_{\beta 0} = \{(b, b)\} \\
C_\gamma &= C_{\gamma 0} \cup C_{\gamma 2} \text{ where } C_{\gamma 0} = \{e\} \text{ and } C_{\gamma 2} = \{(b, b)\} \\
C_\delta &= C_{\delta 0} \cup C_{\delta 2} \text{ where } C_{\delta 0} = \{e\} \text{ and } C_{\delta 2} = \{(b, b)\} \\
C_\rho &= C_{\rho 0} \cup C_{\rho 2} \text{ where } C_{\rho 0} = \{a\} \text{ and } C_{\rho 2} = \{(b, b)\} \\
C_\eta &= C_{\eta 0} = \{e\}
\end{align*}
\[ C_e = C_{\theta 2} = [(a, b)] \]
\[ C_{\mu} = C_{\mu 0} \cup C_{\mu 2} \text{ where } C_{\mu 0} = [e] \text{ and } C_{\mu 2} = [(b, b)] \]

Step 3:
\[ C_{\gamma 0} = C_{\alpha 01} = \phi \]
\[ C_{\gamma 2} = C_{\beta 21} = [(r_1, r_2) \mid r_1 \in B, r_2 \in B] \]
where \( B \) is the sublanguage of trees \([b]\)

\[ C_{\sigma 0} = C_{\delta 01} = [e] \]
\[ C_{\sigma 2} = C_{\delta 21} = [(r_1, r_2) \mid r_1 \in B, r_2 \in B] \]
\[ C_{\tau 0} = C_{\delta 01} = [e] \]
\[ C_{\tau 2} = C_{\delta 21} = [(r_1, r_2) \mid r_1 \in A, r_2 \in B] \]

where \( A \) is the sublanguage of trees \([a]\)

Now the nonterminals and their equivalent sub-languages are enumerated:

<table>
<thead>
<tr>
<th>Nonterminal</th>
<th>Sublanguage</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(\phi)</td>
</tr>
<tr>
<td>A</td>
<td>([a])</td>
</tr>
<tr>
<td>B</td>
<td>([b])</td>
</tr>
<tr>
<td>E</td>
<td>([e])</td>
</tr>
</tbody>
</table>

Now the grammar rules can be constructed:

From the relation \( \beta = \alpha S \):

\[ S \rightarrow S \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow b \quad B \]

Note:
1. \( S \) is the nonterminal corresponding to the sublanguage, \( \phi \).
2. \( C_{\alpha 01} \) has the sublanguage \( \phi \) concatenated at its 1st position. (i.e., there are no trees of depth 0 in the training set.)
3. \( \beta = \alpha S \) is a tree with \( S \) concatenated in the 1st position.
4. \( C_{\beta 21} \) is specified by the sublanguages \([b]\) and \([b]\), respectively.
5. \( B \) and \( B \) are the nonterminals corresponding to \([b]\) and \([b]\).

From the relation \( \gamma = \beta b \):

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow b \quad B \]

Note:
1. \( B \) is the nonterminal corresponding to the sublanguage, \([b]\).
2. \( C_{\beta 21} \) has \( [b] \) in the 1st position (at depth 2).
3. \( \gamma = \beta b \) is a tree with \( b \) concatenated in the 1st position.
4. \( C_{\gamma 21} \) is specified by the sublanguages \([b]\) and \([b]\), respectively.
5. \( B \) and \( B \) are the nonterminals corresponding to \([b]\) and \([b]\).

Also \( B \rightarrow b \) because \( S \) is in the training set.

\[ B \rightarrow b \]

Now consider the relation \( \theta = \gamma ab \)

This yields the rule:

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow A \quad B \quad A \quad B \]

Because 1. \( B \) is the nonterminal corresponding to \([b]\).
2. \( C_{\beta 21} \) has \([b]\) in its 1st position.
3. \( \theta = \gamma ab \) is a tree with \( b \) concatenated in its 2nd position.
4. \( C_{\beta 21} \) is specified by the sublanguages \([a]\) and \([b]\), respectively.
5. \( A \) and \( B \) are the nonterminals corresponding to \([a]\) and \([b]\), respectively.

The relation \( \lambda = \theta ab \) yields

\[ A \rightarrow a \]

because \( \lambda \) is in the training set.

Further, tests with subtrees from the training set will show that all the rules have now been found.

The entire production set is shown below:

\[ S \rightarrow S \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow B \quad B \]

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow A \quad B \quad A \quad B \]

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow A \quad B \quad A \quad B \]

Note: This grammar generates all of the samples in the training set and in fact generates a language larger than the real one. For example, this grammar would generate the following trees which are not in the real language:

\[ B \rightarrow b \]

Similarly, for \( k=2 \), we have the inferred production set

\[ S \rightarrow S \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow B \quad B \]

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow A \quad C \]

\[ S \rightarrow S \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow B \quad B \]

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow A \quad C \]

\[ S \rightarrow S \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow B \quad B \]

\[ B \rightarrow b \]
\[ \quad \rightarrow \]
\[ \quad \quad \rightarrow A \quad C \]
The language generated by this grammar is exactly that generated by the true grammar.

For $k=5$, we have the production set

\[
\begin{align*}
S & \rightarrow S \\
E & \rightarrow b \\
C & \rightarrow b \\
A & \rightarrow a \\
B & \rightarrow b \\
A & \rightarrow b \\
C & \rightarrow b \\
D & \rightarrow b
\end{align*}
\]

The language generated by this grammar is exactly the training set.

The tree grammar inference methods of Bhar- 
gava and Fu (6) and Gonzalez and Thomason (7) 
are similar in that they both assume recursiveness 
whenever there is the slightest evidence of it. It is 
in this sense that they are similar to the k-tail 
method with $k=1$. In the k-tail method, when $k=1$, 
the “loosest” nontrivial grammar is produced. In 
many cases, this will be the same grammar as pro-
duced by both methods. The k-tail method will 
produce more satisfactory grammars when $k > 1$ 
and when the training set is of adequate size.

**AN INFERENCE PROCEDURE FOR WEB GRAMMARS**

In his work on language identification in the 
limit, Gold (8) noted the importance of correctly 
ordering the information sequence. Most other 
grammatical inference researchers have also noted 
this importance. An interesting demonstration of 
the need to carefully select the training sequence is 
the work by Winston (9). The purpose of the work 
was to develop a system which could learn structural 
descriptions of scenes by analyzing specially selected 
examples. This work is now formalized and related 
to the grammar inference problem.

The basic idea will be to correlate the derivation 
diagram of a web grammar with the semantic 
net used by Winston. They by following the steps 
used by Winston on the semantic net and finding 
equivalent steps for the derivation diagram, 
the method can be translated into web grammar 
terminology. The result will be a grammatical inference 
procedure for web grammars which can be applied 
more generally than in the specific block world con-
sidered. A brief review of the derivation diagram 
of web grammars (10) will be required to support 
this discussion.

1. **The Derivation Diagram of Context-Free Web Grammars**

Study of the context-free class of web languages 
reveals that many of the formal language properties 
of string language also hold for the corresponding 
web languages. One example is the existence for 
context-free web grammars of a structure similar 
to a derivation tree for context-free string grammars 
(10). The definition of this structure, called a 
derivation diagram is now given and an example is 
given in Figure 1.

A new, unique relation called the direct des-
cendant relation is introduced. For a pair of nodes 
$(n_1, n_2)$ connected by this relation, $n_2$ is 
called the direct descendant of $n_1$. $n_i$ is called the 
direct ancestor of $n_2$. A node $n_k$ is called a descendant 
of $n_1$ if there is a sequence $n_1, n_2, \ldots n_k$ such that $n_{k+1}$ 
is a direct descendant of $n_i$. $n_1$ is called an ancestor 
of $n_k$.

**Definition 1:**

$D$, a web, is a derivation diagram for a context-
free web grammar $G=(V_N, V_T, P, S)$ if:

1. There is one node called the root with no 
   ancestors whose label is $S$, the start 
   symbol of $G$.
2. All other nodes have exactly one direct 
   ancestor and every node is a descendant 
   of the root.
3. Every node has a label which is a symbol 
   in $V_N$.
4. If a node $n$ has at least one descendant 
   and has label $A$, then $A$ must be in $V_N$.
5. If nodes $n_1, n_2, \ldots, n_k$ are the direct des-
cendants of node $n$ with labels $A_1, A_2, \ldots, A_k$, respectively, $A \rightarrow \beta$ must be a pro-
duction of $P$ of $G$ where $N_1, n_1, n_2, \ldots, n_k$ 
and the $A_i$ is the label of the node $n_i$ in 
$\beta, i=1, \ldots, k$.
6. $n_i$ and $n_j$ are connected by relation $r$ if 
   and only if
   a) one is the direct descendant of the 
      other and $r$ is the direct descendant 
      relation or
   b) $n_i$ and $n_j$ are both direct descendants 
      of $\lambda, A \rightarrow \beta$ is a rule in $P$ 
      and $A_i$ is a subweb of $\beta$ or
   c) $n_i$ and some node $n_k$ are connected 
      by relation $r$ and $n_i$ is the direct 
      descendant of $n_k$ through the rule $A \rightarrow \lambda$ 
      a rule in $P$ and the $r$ between $n_i$ and $n_k$ results from the embedding mapping 
      of $A$.
There are two kinds of subdiagrams which are of interest. The first, called the skeleton of the derivation diagram, is obtained by keeping all nodes and all direct descendant relations and erasing all other relations. The result shown in Figure 1(c) nicely illustrated the basic structure of the derivation.

The second subdiagram of interest is called a section. If \( m_1 \) is a frontier node of the skeleton (i.e., has no descendants), let \( n_0, \ldots, n_k \) be a path to \( m_i \) from the root node, \( n_0 \) along only descendant edges. Let \( m_1, m_2, \ldots, m_k \) be all of the frontier nodes. Then a set \( C \) of nodes of the derivation diagram is a crosscut set if \( C \cap [n_0, \ldots, n_k] \) is a singleton for all \( 1 \leq i \leq k \). A crosscut set, \( C \), together with all of the edges of the derivation diagram between nodes of \( C \) is called a section. Naturally, only those edges are kept which are connected to two nodes which are both kept. A section, illustrated in Figure 1(d), nicely illustrates the basic structure of sentential forms.

2. Interpretation of Winston's System

An example of the type of scene Winston's system analyzes is shown in Figure 2. The sequence of examples Winston found necessary to train the system is shown in Figure 3. Notice that Winston's method uses negative samples in the form of "near misses" as shown in scene 2 and scene 3. The des-

\begin{itemize}
  \item \( S = \) start symbol
  \item \( A = \) Arch
  \item \( B = \) block
  \item \( P = \) pillar
  \item \( C = \) crossbar
  \item \( F = \) front
  \item \( I = \) side
  \item \( T = \) top
  \item \( f = \) in front of
  \item \( u = \) under
  \item \( l = \) left of
\end{itemize}
description that is finally learned is shown in Figure 4. It is assumed that all of the concepts illustrated (except ARCH) have already been learned. Each sample in the training sequence is constructed so that it has only one difference from the already learned description. Scene 2 illustrates that the supports of the arch must not abut. Scene 3 illustrates that A must be supported by B and C. Scene 4 illustrates that a more general object than a BRICK may be used as a top.

The description in Figure 4 can be interpreted as a hierarchical graph model and as a derivation diagram of a web grammar. As such, it can be converted to a web grammar. Some of the rules of this grammar are shown in Figure 5. These rules are created from Figure 5 by generating a rule when a relationship such as "a—kind-of" or "one-part-is" is encountered in the diagram. Thus, the grammar will have a derivation diagram similar to Figure 4. In this case, the system is learning one rule. That is, it is trying to find the predicate which describes the right side of rule (1). If this predicate can be learned, it can then be used to analyze higher order patterns containing it.

Many important nonterminals in a web grammar will not occur in recursive rules. These nonterminals will be important because they represent important semantic concepts which give "meaning" to the structural descriptions. To learn an individual rule in a web grammar, the system must be able to learn the most general description possible for each object on the right-hand side. Assuming the form of the rule is known (this is generally learned from the first sample), then learning the exact rule becomes a matter of finding how much each object may be generalized. In this case, the original description of ARCH might contain the objects A, B, and C; that is, an exact description of this particular scene. This description would be of little general use because no slightly different arch could be identified. Even the appropriate parse of this scene is not known because grammars describing it might be ambiguous.

In a general formalism an object like A is described by properties like orientation and shape. These properties allow successive generalization to occur according to what values of a particular property are important. The structure which describes and systematizes the generalization process is called the property lattice.

Definition 2:
A set of elements \( C = \{c_1, c_2, \ldots\} \) is said to be partially ordered (hierarchical) if there exists a relation \( (\leq) \) defined on the elements of \( C \) which is:

1. Reflexive: \( c \leq c \).
2. Antisymmetric: \( c_1 \leq c_2 \) and \( c_2 \leq c_1 \) implies \( c_2 = c_1 \);
3. Transitive: \( c_1 \leq c_2 \) and \( c_2 \leq c_3 \) implies \( c_1 \leq c_3 \).

If \( C \) is a partially ordered set and \( X \) is any subset of \( C \), then \( a \in X \) is a lower bound of \( X \) if \( a \leq x \) for all \( x \in X \) and \( a \) is an upper bound of \( X \) if \( x \leq a \) for all \( x \in X \). A lower bound \( b \) of \( X \) is called the...

![Figure 2. An Example of an Arch](image)

![Figure 2. An Example of an Arch](image)

![Figure 2. An Example of an Arch](image)

![Figure 2. An Example of an Arch](image)

![Figure 2. An Example of an Arch](image)

![Figure 2. An Example of an Arch](image)

![Figure 2. An Example of an Arch](image)
ARCH

HAS PROPERTY OF

ARCH

MUST BE SUPPORTED BY

ONE-PART-IS

BY

STANDING

B

MUST NOT ABUT

C

ORIENTATION

LEFT-OF

A KIND OF

SPATIAL RELATION

OBJECT

BRICK

A KIND OF

ARCH

(ARCH) (STANDING BRICK) (STANDING BRICK)

WHERE $r_1$ = must be supported by

$r_2$ = left of but must not abut

(1) (ARCH)

(2) (OBJECT) A

(3) (OBJECT) (BRICK)

(4) (OBJECT) (ARCH)

(5) (BRICK) B

(6) (BRICK) C

Figure 4. Derivation of an Arch

Figure 5. Some Web Grammar Rules Describing an Arch
greatest lower bound (g.l.b.) of \( X \) if for every \( a \) that is a lower bound of \( X \), \( a \leq b \). Similarly, an upper bound \( d \) of \( X \) is called the least upper bound if for every \( e \) that is an upper bound of \( X \), \( d \leq e \). A partially ordered set of \( C \) in which any two elements have a least upper bound and a greatest lower bound is called a lattice.

In the case of concept learning here, the elements of \( C \) are called concepts and consist of subsets of samples containing certain property values. The partial order relation considered is set inclusion. The purpose of the learning procedure will be to find that concept which contains all of the samples showing allowed property values and none of the samples having disallowed property values. The procedure to be used in learning a concept will be as follows:

1. Whenever a set of positive samples are given, then all lower bounds of the set in the lattice are allowed as the possible concept. The least upper bound of the set and all its lower bounds are also allowed.
2. Whenever a set of negative samples are given, then all upper bounds of the set in the lattice are disallowed as the possible concept. The greatest lower bound of the set and all its upper bounds are also disallowed.
3. Whenever a new positive sample is given, then the new allowed part of the lattice is the set of all lower bounds of the least upper bound of the new example and the previously learned least upper bound.
4. Whenever a new negative sample is given, then the new disallowed part of the lattice is the set of all upper bounds of the greatest lower bound of the new example and the previously learned greatest lower bound.
5. When all of the points in the lattice are either allowed or disallowed, the correct concept is the least upper bound of the allowed part of the lattice and is said to have been learned.

The purpose of this study will be to see how the lattice can help in selecting a good training set and to see how grammars can help in setting up the lattice. In many practical cases, properties are neither all independent nor all dependent. In these cases, the property lattice is more uneven. Fortunately, the property lattice can be constructed from the grammar if the grammar is in the right form as is shown in Figure 6. Note in this case that a (STANDING TRIANGULAR PRISM) is not allowed by the grammar so the higher order concepts (STANDING) and (TRIANGULAR PRISM) are also not present. How, the number and selection of samples necessary to learn a concept in this lattice can be investigated. To generalize to the concept (PRISM), 2 positive samples (STANDING BRICK) and (LYING TRIA PRISM) must be given. To generalize only to (BRICK) or (LYING), all three samples (2 positive and 1 negative) must be given. To generalize to (STANDING BRICK) only, two samples must be given.

Thus, by using the grammatically formalism, Winston's procedure, as just formalized, can be stated as follows:

1. Assume that a given set of properties and predicate forms are known to be appropriate from a priori information about the application.
2. Given a sample, get all possible parses of it with these forms and arrange the parse non-terminals in a property lattice.
3. Then, by giving a sequence of appropriate positive and negative samples, and using least upper and greatest lower bound operations in the lattice, converge to the correct parse common to all positive samples and including no negative samples.
4. Construct the grammar rule reflecting this parse. An example of applying this procedure to a Winston-like problem is now given.

Given the Grammon:

\[
\begin{align*}
\text{(PRISM)} & \rightarrow \text{(BRICK)} \\
\text{(PRISM)} & \rightarrow \text{(LYING)} \\
\text{(BRICK)} & \rightarrow \text{(LYING BRICK)} \\
\text{(BRICK)} & \rightarrow \text{(STANDING BRICK)} \\
\text{(LYING)} & \rightarrow \text{(LYING BRICK)} \\
\text{(LYING)} & \rightarrow \text{(LYING TRIA PRISM)} \\
\end{align*}
\]

Figure 6. A Lattice Constructed From A Grammar
Example 2:

Assume we are given a problem in which the only objects are rectangular prisms and the only properties detectable are size, shape, and color. Furthermore, assume that green cubes do not exist. A lattice illustrating these properties is shown in Figure 7. The objects, properties, and relations are summarized below in Table 1.

<table>
<thead>
<tr>
<th>Object</th>
<th>Properties</th>
<th>Values</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>Size</td>
<td>Larger</td>
<td>Supported by</td>
</tr>
<tr>
<td>Prisms</td>
<td></td>
<td>Smaller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shape</td>
<td>Rectangular</td>
<td>Same color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prism</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>Red</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Green</td>
<td></td>
</tr>
</tbody>
</table>

We now wish to learn the concept of a pyramid. For illustrative purposes it is assumed that a legal pyramid can have cubes or rectangular prisms but supporting objects can only be red in color. That is only the top object can be green. To being, a positive sample of a pyramid (shown in Figure 8) is presented and the pattern is parsed. The parse or derivation diagram or semantic net resulting is shown in Figure 9.

Now, by presenting an appropriate sequence

Object Lattice

P = (P1, P2)

P1 = COLOR, 0 = RED, 1 = GREEN
P2 = SHAPE, 0 = CUBE, 1 = RECTANGLE

of positive and negative samples, the teacher must illustrate the most general object or relation which is allowed in each position. This example will concentrate just on the objects and for the moment ignore the fact that the relations must be learned also. The supporting objects in the pyramid can be any shape but must be red. This is illustrated by the (00,01) entry in the lattice. This can be illustrated by three samples: 00 and 01 as positive samples and 11 as a negative sample. The top object can be green. Since a cube cannot be green, this is illustrated by the (00,01,11) entry in the lattice. This state in the lattice can be learned by presenting 00, 01, and 11 as positive samples. Thus, for each individual object, three samples must be given. But, since these can occur in various combinations with the other objects, a total of 27 combinations must be presented to completely learn the definition of the pyramid. The samples are shown in Figure 10. Note that if the objects can be considered independent only seven samples need to be given. These same are shown with asterisks in Figure 10.

The derivation diagram which is finally learned is shown in Figure 11. The grammar rule learned is extracted from this diagram by putting the ancestor of the “One-part-is” relation on the left-hand side and the descendants on the right-hand side. This rule is shown in Figure 12. The embedding of this rule is somewhat arbitrary.

Several conclusions can be drawn from this example. First, if there are several properties involved and these properties take on several values and it is necessary to learn a pattern containing several objects, then many samples must be used in training unless some heuristic assumption is made. Second, if one part of the pattern can be assumed independent of
inference (or the “goodness of fit”) and the applicability to real-world problems. A proposal for inferring web grammar from pictorial patterns can be found in [10].

REFERENCES


CONCLUSIONS AND REMARKS

This paper presents some preliminary results in the inference of tree and web grammars. It is hoped that the preliminary results will stimulate new and better inference methods for high-dimensional grammars, particularly concerning the quality of

other parts, the number of samples needed to learn it can be greatly reduced. Third, this method as shown does not specify the embedding.

Figure 10. Training Samples

Figure 11. Final Derivation Diagram

Figure 12. The Resulting Grammar Rule