CHARACTERIZING STRUCTURAL DESCRIPTIONS PRODUCED BY VARIOUS GRAMMATICAL FORMALISMS*

K. Vijay-Shanker David J. Weir Aravind K. Joshi

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa 19104

ABSTRACT

We consider the structural descriptions produced by various grammatical formalisms in terms of the complexity of the paths and the relationship between paths in the sets of structural descriptions that each system can generate. In considering the relationship between formalisms, we show that it is useful to abstract away from the details of the formalism, and examine the nature of their derivation process as reflected by properties of their *derivation trees*. We find that several of the formalisms considered can be seen as being closely related since they have derivation tree sets with the same structure as those produced by Context-Free Grammars. On the basis of this observation, we describe a class of formalisms which we call Linear Context-Free Rewriting Systems, and show they are recognizable in polynomial time and generate only semilinear languages.

1 Introduction

Much of the study of grammatical systems in computational linguistics has been focused on the weak generative capacity of grammatical formalism. Little attention, however, has been paid to the structural descriptions that these formalisms can assign to strings, i.e. their strong generative capacity. This aspect of the formalism is both linguistically and computationally important. For example, Gazdar (1985) discusses the applicability of Indexed Grammars (IG's) to Natural Language in terms of the structural descriptions assigned; and Berwick (1984) discusses the strong generative capacity of Lexical-Functional Grammar (LFG) and Government and Bindings grammars (GB). The work of Thatcher (1973) and Rounds (1969) define formal systems that generate tree sets that are related to CFG's and IG's.

We consider properties of the tree sets generated by CFG's, Tree Adjoining Grammars (TAG's), Head Grammars (HG's), Categorial Grammars (CG's), and IG's. We examine both the complexity of the paths of trees in the tree sets, and the kinds of dependencies that the formalisms can impose between paths. These two properties of the tree sets are not only linguistically relevant, but also have computational importance. By considering derivation trees, and thus abstracting away from the details of the composition operation and the structures being manipulated, we are able to state the similarities and differences between the formalisms. It is striking that from this point of view many formalisms can be grouped together as having identically structured derivation tree sets. This suggests that by generalizing the notion of context-freeness in CFG's, we can define a class of grammatical formalisms that manipulate more complex structures. In this paper, we outline how such family of formalisms can be defined, and show that like CFG's, each member possesses a number of desirable linguistic and computational properties: in particular, the constant growth property and polynomial recognizability.

*This work was partially supported by NSF grants MCS-82-19116-CER, MCS-82-07294 and DCR-84-10413, ARO grant DAA 29-84-9-0027, and DARPA grant N00014-85-K0018. We are very grateful to Tony Kroch, Michael Palis, Sunil Shende, and Mark Steedman for valuable discussions.

2 Tree Sets of Various Formalisms

2.1 Context-Free Grammars

From Thatcher's (1973) work, it is obvious that the complexity of the set of paths from root to frontier of trees in a local set (the tree set of a CFG) is regular[1]. We define the path set of a tree γ as the set of strings that label a path from the root to frontier of γ. The path set of a tree set is the union of the path sets of trees in that tree set. It can be easily shown from Thatcher's result that the path set of every local set is a regular set. As a result, CFG's can not provide the structural descriptions in which there are nested dependencies between symbols labelling a path. For example, CFG's cannot produce trees of the form shown in Figure 1 in which there are nested dependencies between S and NP nodes appearing on the spine of the tree. Gazdar (1985) argues this is the appropriate analysis of unbounded dependencies in the hypothetical Scandinavian language Norwedish. He also argues that paired English complementizers may also require structural descriptions whose path sets have nested dependencies.

2.2 Head Grammars and Generalized CFG's

Head Grammars (HG's), introduced by Pollard (1984), is a formalism that manipulates headed strings: i.e., strings, one of whose symbols is distinguished as the head. Not only is concatenation of these strings possible, but *head wrapping* can be used to split a string and wrap it around another string. The productions of HG's are very similar to those of CFG's except that the operation used must be made explicit. Thus, the tree sets generated by HG's are similar to those of CFG's, with each node annotated by the operation (concatenation or wrapping) used to combine the headed strings derived by the daughters of

[1] Thatcher actually characterized recognizable sets: for the purposes of this paper we do not distinguish them from local sets.

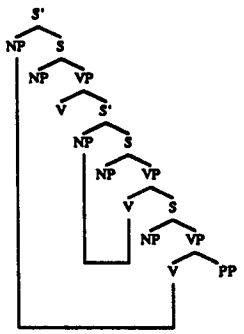

Figure 1: Nested dependencies in Norwedish

that node. A derivation tree giving an analysis of Dutch subordinate clauses is given in Figure 2.

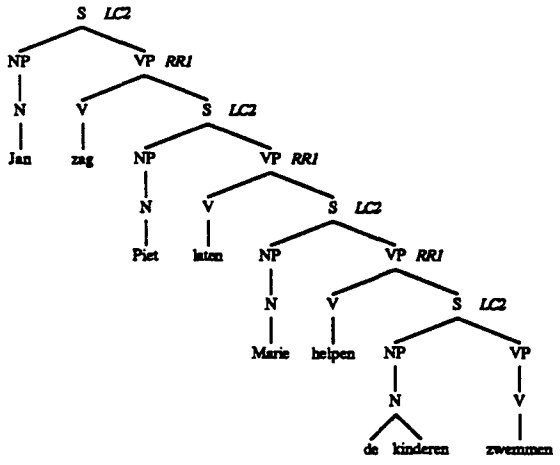

Figure 2: HG analysis of Dutch subordinate clauses

HG's are a special case of a class of formalisms called Generalized Context-Free Grammars, also introduced by Pollard (1984). A formalism in this class is defined by a finite set of operations (of which concatenation and wrapping are two possibilities). As in the case of HG's the annotated tree sets for these formalisms have the same structure as local sets.

2.3 Tree Adjoining Grammars

Tree Adjoining Grammars, a tree rewriting formalism, was introduced by Joshi, Levy and Takahashi (1975) and Joshi (1983/85). A TAG consists of a finite set of *elementary* trees that are either *initial* trees or *auxiliary* trees. Trees are composed using an operation called *adjoining*, which is defined as follows. Let η be some node labeled X in a tree γ (see Figure 3). Let γ' be a tree with root and foot labeled by X. When γ' is adjoined at η in the tree γ we obtain a tree γ''. The subtree under η is excised from γ, the tree γ' is inserted in its place and the excised subtree is inserted below the foot of γ'.

It can be shown that the path set of the tree set generated by a TAG G is a context-free language. TAG's can be used to give

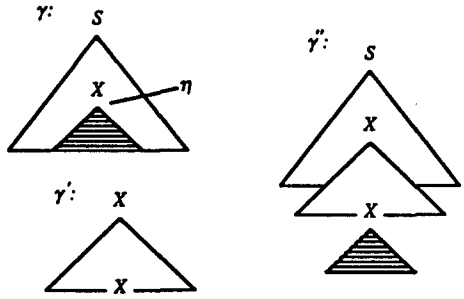

Figure 3: Adjunction operation

the structural descriptions discussed by Gazdar (1985) for the unbounded nested dependencies in Norwedish, for cross serial dependencies in Dutch subordinate clauses, and for the nestings of paired English complementizers.

From the definition of TAG's, it follows that the choice of adjunction is not dependent on the history of the derivation. Like CFG's, the choice is predetermined by a finite number of rules encapsulated in the grammar. Thus, the *derivation trees* for TAG's have the same structure as local sets. As with HG's derivation structures are annotated; in the case of TAG's, by the trees used for adjunction and addresses of nodes of the elementary tree where adjunctions occurred.

We can define derivation trees inductively on the length of the derivation of a tree γ. If γ is an elementary tree, the derivation tree consists of a single node labeled γ. Suppose γ results from the adjunction of $\gamma_1, \ldots, \gamma_k$ at the k distinct tree addresses $n_1, \ldots, n_k$ in some elementary tree γ', respectively. The tree denoting this derivation of γ is rooted with a node labeled γ' having k subtrees for the derivations of $\gamma_1, \ldots, \gamma_k$. The edge from the root to the subtree for the derivation of γ_i is labeled by the address n_i. To show that the derivation tree set of a TAG is a local set, nodes are labeled by pairs consisting of the name of an elementary tree and the address at which it was adjoined, instead of labelling edges with addresses. The following rule corresponds to the above derivation, where $\gamma_1, \ldots, \gamma_k$ are derived from the auxiliary trees $\beta_1, \ldots, \beta_k$, respectively.

$$\langle \gamma', n \rangle \rightarrow \langle \beta_1, n_1 \rangle \ldots \langle \beta_k, n_k \rangle$$

for all addresses n in some elementary tree at which γ' can be adjoined. If γ' is an initial tree we do not include an address on the left-hand side.

2.4 Indexed Grammars

There has been recent interest in the application of Indexed Grammars (IG's) to natural languages. Gazdar (1985) considers a number of linguistic analyses which IG's (but not CFG's) can make, for example, the Norwedish example shown in Figure 1. The work of Rounds (1969) shows that the path sets of trees derived by IG's (like those of TAG's) are context-free languages. Trees derived by IG's exhibit a property that is not exhibited by the trees sets derived by TAG's or CFG's. Informally, two or more paths can be dependent on each other: for example, they could be required to be of equal length as in the trees in Figure 4.

IG's can generate trees with dependent paths as in Figure 4b. Although the path set for trees in Figure 4a is regular, no CFG

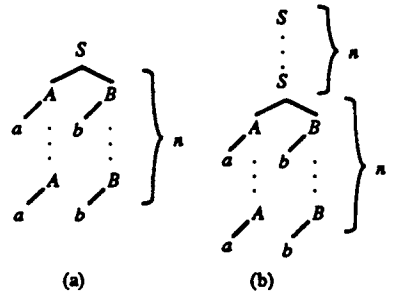

Figure 4: Example with dependent paths

generates such a tree set. We focus on this difference between the tree sets of CFG's and IG's, and formalize the notion of dependence between paths in a tree set in Section 3.

An IG can be viewed as a CFG in which each nonterminal is associated with a stack. Each production can push or pop symbols on the stack as can be seen in the following productions that generate tree of the form shown in Figure 4b.

$S(\alpha) \rightarrow S(\eta\alpha)$	push η
$S(\alpha) \rightarrow A(\alpha)B(\alpha)$	share stack
$A(\eta\alpha) \rightarrow aA(\alpha)$	pop η
$B(\eta\alpha) \rightarrow bB(\alpha)$	pop η
$A() \rightarrow a$	
$B() \rightarrow b$	

Gazdar (1985) argues that sharing of stacks can be used to give analyses for coordination. Analogous to the sharing of stacks in IG's, Lexical-Functional Grammar's (LFG's) use the unification of unbounded hierarchical structures. Unification is used in LFG's to produce structures having two dependent spines of unbounded length as in Figure 5. Bresnan, Kaplan, Peters, and Zaenen (1982) argue that these structures are needed to describe crossed-serial dependencies in Dutch subordinate clauses. Gazdar (1985) considers a restriction of IG's in which no more

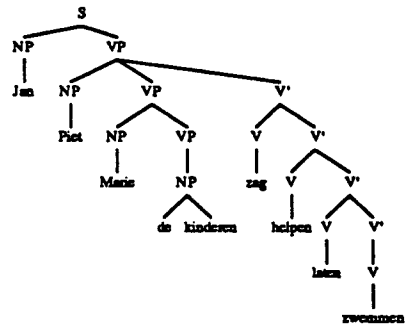

Figure 5: LFG analysis of Dutch subordinate clauses

than one nonterminal on the right-hand-side of a production can inherit the stack from the left-hand-side. Unbounded dependencies between branches are not possible in such a system. TAG's can be shown to be equivalent to this restricted system. Thus, TAG's can not give analyses in which dependencies between arbitrarily large branches exist.

2.5 Categorial Grammars

Steedman (1986) considers Categorial Grammars in which both the operations of function application and composition may be used, and in which function can specify whether they take their arguments from their right or left. While the generative power of CG's is greater that of CFG's, it appears to be highly constrained. Hence, their relationship to formalisms such as HG's and TAG's is of interest. On the one hand, the definition of composition in Steedman (1985), which technically permits composition of functions with unbounded number of arguments, generates tree sets with dependent paths such as those shown in Figure 6. This kind of dependency arises from the use of the

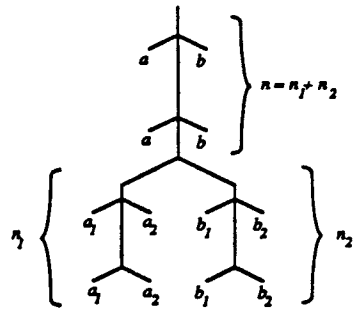

Figure 6: Dependent branches from Categorial Grammars

composition operation to compose two arbitrarily large categories. This allows an unbounded amount of information about two separate paths (e.g. an encoding of their length) to be combined and used to influence the later derivation. A consequence of the ability to generate tree sets with this property is that CG's under this definition can generate the following language which can not be generated by either TAG's or HG's.

$$\{ a^n a_1^{n_1} a_2^{n_1} b_1^{n_2} b_2^{n_2} b^n \mid n = n_1 + n_2 \}$$

On the other hand, no linguistic use is made of this general form of composition and Steedman (personal communication) and Steedman (1986) argues that a more limited definition of composition is more natural. With this restriction the resulting tree sets will have independent paths. The equivalence of CG's with this restriction to TAG's and HG's is, however, still an open problem.

2.6 Multicomponent TAG's

An extension of the TAG system was introduced by Joshi et al. (1975) and later redefined by Joshi (1987) in which the adjunction operation is defined on sets of elementary trees rather than single trees. A multicomponent Tree Adjoining Grammar (MC-TAG) consists of a finite set of finite elementary tree sets. We must adjoin all trees in an auxiliary tree set together as a single step in the derivation. The adjunction operation with respect to tree sets (multicomponent adjunction) is defined as follows.

Each member of a set of trees can be adjoined into distinct nodes of trees in a *single elementary tree set*, i.e, derivations always involve the adjunction of a derived auxiliary tree set into an elementary tree set.

Like CFG's, TAG's, and HG's the *derivation* tree set of a MCTAG will be a local set. The derivation trees of a MCTAG are similar to those of a TAG. Instead of the names of elementary trees of a TAG, the nodes are labeled by a sequence of names of trees in an elementary tree set. Since trees in a tree set are adjoined together, the addressing scheme uses a sequence of pairings of the address and name of the elementary tree adjoined at that address. The following context-free production captures the derivation step of the grammar shown in Figure 7, in which the trees in the auxiliary tree set are adjoined into themselves at the root node (address ϵ).

$$\langle\langle\beta_1,\beta_2,\beta_3\rangle,\bar{\eta}\rangle \rightarrow \langle\langle\beta_1,\beta_2,\beta_3\rangle,\langle\langle\beta_1,\epsilon\rangle,\langle\beta_2,\epsilon\rangle,\langle\beta_3,\epsilon\rangle\rangle\rangle$$

The path complexity of the tree set generated by a MCTAG is not necessarily context-free. Like the string languages of MCTAG's, the complexity of the path set increases as the cardinality of the elementary tree sets increases, though both the string languages and path sets will always be semilinear.

MCTAG's are able to generate tree sets having dependent paths. For example, the MCTAG shown in Figure 7 generates trees of the form shown in Figure 4b. The number of paths that

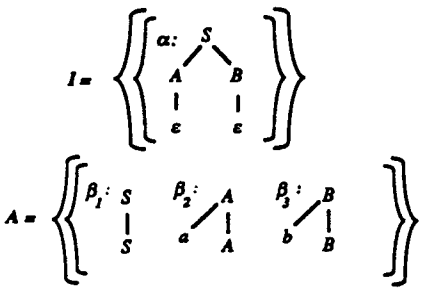

Figure 7: A MCTAG with dependent paths

can be dependent is bounded by the grammar (in fact the maximum cardinality of a tree set determines this bound). Hence, trees shown in Figure 8 can not be generated by any MCTAG (but can be generated by an IG) because the number of pairs of dependent paths grows with n.

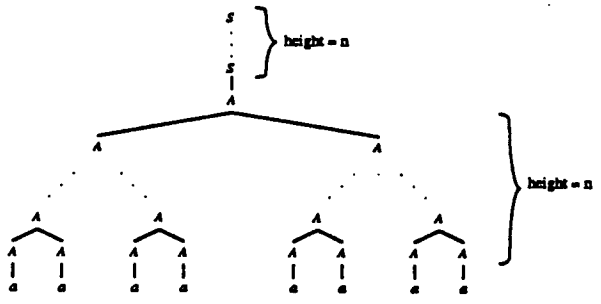

Figure 8: Trees with unbounded dependencies

Since the derivation trees of TAG's, MCTAG's, and HG's are local sets, the choice of the structure used at each point in a derivation in these systems does not depend on the context at that point within the derivation. Thus, as in CFG's, at any point in the derivation, the set of structures that can be applied is determined only by a finite set of rules encapsulated by the grammar. We characterize a class of formalisms that have this property in Section 4. We loosely describe the class of all such systems as Linear Context-Free Rewriting Formalisms. As is described in Section 4, the property of having a derivation tree set that is a local set appears to be useful in showing important properties of the languages generated by the formalisms. The semilinearity of Tree Adjoining Languages (TAL's), MCTAL's, and Head Languages (HL's) can be proved using this property, with suitable restrictions on the composition operations.

3 Dependencies between Paths

Roughly speaking, we say that a tree set contains trees with dependent paths if there are two paths $p_\gamma = u_\gamma v_\gamma$ and $q_\gamma = u_\gamma w_\gamma$ in each $\gamma \in \Gamma$ such that u_γ is some, possibly empty, shared initial subpath; v_γ and w_γ are not bounded in length; and there is some "dependence" (such as equal length) between the set of all v_γ and w_γ for each $\gamma \in \Gamma$. A tree set may be said to have dependencies between paths if some "appropriate" subset can be shown to have dependent paths as defined above.

We attempt to formalize this notion in terms of the tree pumping lemma which can be used to show that a tree set does *not* have dependent paths. Thatcher (1973) describes a tree pumping lemma for recognizable sets related to the string pumping lemma for regular sets. The tree in Figure 9a can be denoted by $t_1 t_2 t_3$ where tree substitution is used instead of concatenation. The tree pumping lemma states that if there is tree, $t = t_1 t_2 t_3$, generated by a CFG G, whose height is more than a predetermined bound k, then all trees of the form $t_1 t_2^i t_3$ for each $i \geq 0$ will also generated by G (as shown in Figure 9b). The string pumping lemma for CFG's ($uvwxy$-theorem) can be seen as a corollary of this lemma.

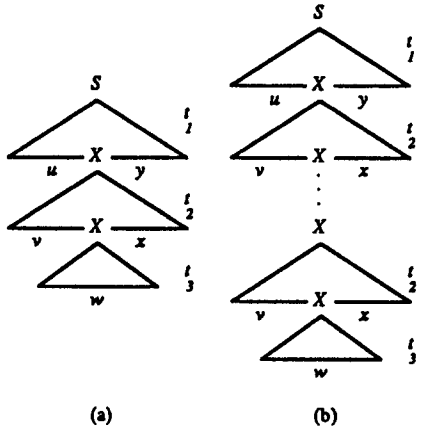

Figure 9: Tree pumping lemma for local sets

The fact that local sets do not have dependent paths follows

from this pumping lemma: a single path can be pumped independently. For example, let us consider a tree set containing trees of the form shown in Figure 4a. The tree t_2 must be on one of the two branches. Pumping t_2 will change only one branch and leave the other branch unaffected. Hence, the resulting trees will no longer have two branches of equal size.

We can give a tree pumping lemma for TAG's by adapting the $uvwxy$-theorem for CFL's since the tree sets of TAG's have *independent* and *context-free* paths. This pumping lemma states that if there is tree, $t = t_1t_2t_3t_4t_5$, generated by a TAG G, such that its height is more than a predetermined bound k, then all trees of the form $t_1t_2^it_3t_4^it_5$ for each $i \geq 0$ will also generated by G. Similarly, for tree sets with independent paths and more complex path sets, tree pumping lemmas can be given. We adapt the string pumping lemma for the class of languages corresponding to the complexity of the path set.

A geometrical progression of language families defined by Weir (1987) involves tree sets with increasingly complex path sets. The independence of paths in the tree sets of the k^{th} grammatical formalism in this hierarchy can be shown by means of tree pumping lemma of the form $t_1t_2^it_3t_4^i \ldots t_{2^k+1}^it_{2^k+1+1}$. The path set of tree sets at level $k+1$ have the complexity of the string language of level k.

The independence of paths in a tree set appears to be an important property. A formalism generating tree sets with complex path sets can still generate only semilinear languages if its tree sets have independent paths, and semilinear path sets. For example, the formalisms in the hierarchy described above generate semilinear languages although their path sets become increasingly more complex as one moves up the hierarchy. From the point of view of recognition, independent paths in the derivation structures suggests that a top-down parser (for example) can work on each branch independently, which may lead to efficient parsing using an algorithm based on the *Divide and Conquer* technique.

4 Linear Context-Free Rewriting Systems

From the discussion so far it is clear that a number of formalisms involve some type of context-free rewriting (they have derivation trees that are local sets). Our goal is to define a class of formal systems, and show that any member of this class will possess certain attractive properties. In the remainder of the paper, we outline how a class of Linear Context-Free Rewriting Systems (LCFRS's) may be defined and sketch how semilinearity and polynomial recognition of these systems follows.

4.1 Definition

In defining LCFRS's, we hope to generalize the definition of CFG's to formalisms manipulating *any* structure, e.g. strings, trees, or graphs. To be a member of LCFRS a formalism must satisfy two restrictions. First, any grammar must involve a finite number of elementary structures, composed using a finite number of composition operations. These operations, as we see below, are restricted to be *size* preserving (as in the case of concatenation in CFG) which implies that they will be *linear* and *non-erasing*. A second restriction on the formalisms is that choices during the derivation are independent of the context in the derivation. As will be obvious later, their derivation tree sets will be local sets as are those of CFG's.

Each derivation of a grammar can be represented by a generalized context-free derivation tree. These derivation trees show how the composition operations were used to derive the final structures from elementary structures. Nodes are annotated by the name of the composition operation used at that step in the derivation. As in the case of the derivation trees of CFG's, nodes are labeled by a member of some finite set of symbols (perhaps only implicit in the grammar as in TAG's) used to denote derived structures. Frontier nodes are annotated by zero arity functions corresponding to elementary structures. Each treelet (an internal node with all its children) represents the use of a rule that is encapsulated by the grammar. The grammar encapsulates (either explicitly or implicitly) a finite number of rules that can be written as follows:

$$A \rightarrow f_p(A_1, \ldots, A_n) \qquad n \geq 0$$

In the case of CFG's, for each production

$$p = A \rightarrow u_1A_1 \ldots u_nA_nu_{n+1}$$

(where u_i is a string of terminals) the function f_p is defined as follows.

$$f_p(x_1, \ldots, x_n) = u_1x_1 \ldots u_nx_nu_{n+1}$$

In the case of TAG's, a derivation step in which the derived trees $\bar{\beta}_1, \ldots, \bar{\beta}_n$ are adjoined into β at the addresses $i_1, \ldots, i_n$, would involve the use of the following rule[2].

$$\bar{\beta} \rightarrow A_{\beta,i_1,\ldots,i_n}(\bar{\beta}_1, \ldots, \bar{\beta}_n)$$

The composition operations in the case of CFG's are parameterized by the productions. In TAG's the elementary tree and addresses where adjunction takes place are used to instantiate the operation.

To show that the derivation trees of any grammar in LCFRS is a *local* set, we can rewrite the annotated derivation trees such that every node is labelled by a pair to include the composition operations. These systems are similar to those described by Pollard (1984) as Generalized Context-Free Grammars (GCFG's). Unlike GCFG's, however, the composition operations of LCFRS's are restricted to be *linear* (do not duplicate unboundedly large structures) and *nonerasing* (do not erase unbounded structures, a restriction made in most modern transformational grammars). These two restrictions impose the constraint that the result of composing any two structures should be a structure whose "size" is the sum of its constituents plus some constant. For example, the operation f_p discussed in the case of CFG's (in Section 4.1) adds the constant equal to the sum of the length of the strings $u_1, \ldots, u_{n+1}$.

Since we are considering formalisms with arbitrary structures it is difficult to precisely specify all of the restrictions on the composition operations that we believe would appropriately generalize the concatenation operation for the particular

[2] We denote a tree derived from the elementary tree γ by the symbol $\bar{\gamma}$.

structures used by the formalism. In considering recognition of LCFRS's, we make further assumption concerning the contribution of each structure to the input string, and how the composition operations combine structures in this respect. We can show that languages generated by LCFRS's are semilinear as long as the composition operation does not remove any terminal symbols from its arguments.

4.2 Semilinearity of LCFRL's

Semilinearity and the closely related constant growth property (a consequence of semilinearity) have been discussed in the context of grammars for natural languages by Joshi (1983/85) and Berwick and Weinberg (1984). Roughly speaking, a language, L, has the property of semilinearity if the number of occurrences of each symbol in any string is a linear combination of the occurrences of these symbols in some fixed finite set of strings. Thus, the length of any string in L is a linear combination of the length of strings in some fixed finite subset of L, and thus L is said to have the constant growth property. Although this property is not structural, it depends on the structural property that sentences can be built from a finite set of clauses of bounded structure as noted by Joshi (1983/85).

The property of semilinearity is concerned only with the occurrence of symbols in strings and not their order. Thus, any language that is letter equivalent to a semilinear language is also semilinear. Two strings are letter equivalent if they contain equal number of occurrences of each terminal symbol, and two languages are letter equivalent if every string in one language is letter equivalent to a string in the other language and vice-versa. Since every CFL is known to be semilinear (Parikh, 1966), in order to show semilinearity of some language, we need only show the existence of a letter equivalent CFL.

Our definition of LCFRS's insists that the composition operations are linear and nonerasing. Hence, the terminal symbols appearing in the structures that are composed are not lost (though a constant number of new symbols may be introduced). If $\psi(A)$ gives the number of occurrences of each terminal in the structure named by A, then, given the constraints imposed on the formalism, for each rule $A \rightarrow f_p(A_1, \ldots, A_n)$ we have the equality

$$\psi(A) = \psi(A_1) + \ldots + \psi(A_n) + c_p$$

where c_p is some constant. We can obtain a letter equivalent CFL defined by a CFG in which the for each rule as above, we have the production $A \rightarrow A_1 \ldots A_n u_p$ where $\psi(u_p) = c_p$. Thus, the language generated by a grammar of a LCFRS is semilinear.

4.3 Recognition of LCFRL's

We now turn our attention to the recognition of string languages generated by these formalisms (LCFRL's). As suggested at the end of Section 3, the restrictions that have been specified in the definition of LCFRS's suggest that they can be efficiently recognized. In this section for the purposes of showing that polynomial time recognition is possible, we make the additional restriction that the contribution of a derived structure to the input string can be specified by a bounded sequence of substrings of the input. Since each composition operation is linear and nonerasing, a bounded sequences of substrings associated with the resulting structure is obtained by combining the substrings in each of its arguments using only the concatenation operation, including each substring exactly once. CFG's, TAG's, MCTAG's and HG's are all members of this class since they satisfy these restrictions.

Giving a recognition algorithm for LCFRL's involves describing the substrings of the input that are spanned by the structures derived by the LCFRS's and how the composition operation combines these substrings. For example, in TAG's a derived auxiliary tree spans two substrings (to the left and right of the foot node), and the adjunction operation inserts another substring (spanned by the subtree under the node where adjunction takes place) between them (see Figure 3). We can represent any derived tree of a TAG by the two substrings that appear in its frontier, and then define how the adjunction operation concatenates the substrings. Similarly, for all the LCFRS's, discussed in Section 2, we can define the relationship between a structure and the sequence of substrings it spans, and the effect of the composition operations on sequences of substrings.

A derived structure will be mapped onto a sequence $x_1, \ldots, x_l$ of substrings (not necessarily contiguous in the input), and the composition operations will be mapped onto functions that can defined as follows[3].

$$f(\langle x_1, \ldots, x_{n_1} \rangle, \langle y_1, \ldots, y_{n_2} \rangle) = \langle z_1, \ldots, z_{n_3} \rangle$$

where each z_i is the concatenation of strings from x_j's and y_k's. The linear and nonerasing assumptions about the operations discussed in Section 4.1 require that each x_j and y_k is used exactly once to define the strings $z_1, \ldots, z_{n_3}$. Some of the operations will be constant functions, corresponding to elementary structures, and will be written as $f() = \langle z_1, \ldots z_l \rangle$, where each z_i is a constant, the string of terminal symbols $a_{1,i} \ldots a_{n_i,i}$.

This representation of structures by substrings and the composition operation by its effect on substrings is related to the work of Rounds (1985). Although embedding this version of LCFRS's in the framework of ILFP developed by Rounds (1985) is straightforward, our motivation was to capture properties shared by a family of grammatical systems and generalize them defining a class of related formalisms. This class of formalisms have the properties that their derivation trees are local sets, and manipulate objects, using a finite number of composition operations that use a finite number of symbols. With the additional assumptions, inspired by Rounds (1985), we can show that members of this class can be recognized in polynomial time.

4.3.1 Alternating Turing Machines

We use Alternating Turing Machines (Chandra, Kozen, and Stockmeyer, 1981) to show that polynomial time recognition is possible for the languages discussed in Section 4.3. An ATM has two types of states, existential and universal. In an existential state an ATM behaves like a nondeterministic TM, accepting

[3] In order to simplify the following discussion, we assume that each composition operation is binary. It is easy to generalize to the case of n-ary operations.

if one of the applicable moves leads to acceptance; in an universal state the ATM accepts if all the applicable moves lead to acceptance. An ATM may be thought of as spawning independent processes for each applicable move. A k-tape ATM, M, has a read-only input tape and k read-write work tapes. A *step* of an ATM consists of reading a symbol from each tape and optionally moving each head to the left or right one tape cell. A *configuration* of M consists of a state of the finite control, the nonblank contents of the input tape and k work tapes, and the position of each head. The *space* of a configuration is the sum of the lengths of the nonblank tape contents of the k work tapes. M works in *space* $S(n)$ if for every string that M accepts no configuration exceeds space $S(n)$. It has been shown in (Chandra et al., 1981) that if M works in *space* $\log n$ then there is a deterministic TM which accepts the same language in polynomial time. In the next section, we show how an ATM can accept the strings generated by a grammar in a LCFRS formalism in logspace, and hence show that each family can be recognized in polynomial time.

4.3.2 Recognition by ATM

We define an ATM, M, recognizing a language generated by a grammar, G, having the properties discussed in Section 4.3. It can be seen that M performs a top-down recognition of the input $a_1 \ldots a_n$ in logspace.

The rewrite rules and the definition of the composition operations may be stored in the finite state control since G uses a finite number of them. Suppose M has to determine whether the k substrings $z_1, \ldots, z_k$ can be derived from some symbol A. Since each z_i is a contiguous substring of the input (say $a_{i_1} \ldots a_{i_2}$), and no two substrings overlap, we can represent z_i by the pair of integers $\langle i_1, i_2 \rangle$. We assume that M is in an existential state q_A, with integers i_1 and i_2 representing z_i in the $(2i-1)^{th}$ and $2i^{th}$ work tape, for $1 \leq i \leq k$.

For each rule $p : A \rightarrow f_p(B, C)$ such that f_p is mapped onto the function $\bar{f}_p$ defined by the following rule.

$$\bar{f}_p(\langle x_1, \ldots, x_{n_1} \rangle, \langle y_1, \ldots, y_{n_2} \rangle) = \langle z_1, \ldots, z_k \rangle$$

M breaks $z_1, \ldots, z_k$ into substrings $x_1, \ldots, x_{n_1}$ and $y_1, \ldots, y_{n_2}$ conforming to the definition of $\bar{f}_p$. M spawns as many processes as there are ways of breaking up $z_1, \ldots, z_k$ and rules with A on their left-hand-side. Each spawned process must check if $x_1, \ldots, x_{n_1}$ and $y_1, \ldots, y_{n_2}$ can be derived from B and C, respectively. To do this, the x's and y's are stored in the next $2n_1 + 2n_2$ tapes, and M goes to a universal state. Two processes are spawned requiring B to derive $x_1, \ldots, x_{n_1}$ and C to derive $y_1, \ldots, y_{n_2}$. Thus, for example, one successor process will be have M to be in the existential state q_B with the indices encoding $x_1, \ldots, x_{n_1}$ in the first $2n_1$ tapes.

For rules $p : A \rightarrow f_p()$ such that f_p is constant function, giving an elementary structure, $\bar{f}_p$ is defined such that $\bar{f}_p() = \langle z_1 \ldots z_k \rangle$ where each z is a constant string. M must enter a universal state and check that each of the k constant substrings are in the appropriate place (as determined by the contents of the first $2k$ work tapes) on the input tape. In addition to the tapes required to store the indices, M requires one work tape for splitting the substrings. Thus, the ATM has no more than $6k^{\max} + 1$ work tapes, where $k^{\max}$ is the maximum number of substrings spanned by a derived structure. Since the work tapes store integers (which can be written in binary) that never exceed the size of the input, no configuration has space exceeding $O(\log n)$. Thus, M works in logspace and recognition can be done on a deterministic TM in polynomial tape.

5 Discussion

We have studied the structural descriptions (tree sets) that can be assigned by various grammatical systems, and classified these formalisms on the basis of two features: path complexity; and path independence. We contrasted formalisms such as CFG's, HG's, TAG's and MCTAG's, with formalisms such as IG's and unificational systems such as LFG's and FUG's.

We address the question of whether or not a formalism can generate only structural descriptions with independent paths. This property reflects an important aspect of the underlying linguistic theory associated with the formalism. In a grammar which generates independent paths the derivations of sibling constituents can not share an unbounded amount of information. The importance of this property becomes clear in contrasting theories underlying GPSG (Gazdar, Klein, Pullum, and Sag, 1985), and GB (as described by Berwick, 1984) with those underlying LFG and FUG. It is interesting to note, however, that the ability to produce a bounded number of dependent paths (where two dependent paths can share an unbounded amount of information) does not require machinery as powerful as that used in LFG, FUG and IG's. As illustrated by MCTAG's, it is possible for a formalism to give tree sets with bounded dependent paths while still sharing the constrained rewriting properties of CFG's, HG's, and TAG's.

In order to observe the similarity between these constrained systems, it is crucial to abstract away from the details of the structures and operations used by the system. The similarities become apparent when they are studied at the level of *derivation structures*: derivation tree sets of CFG's, HG's, TAG's, and MCTAG's are all local sets. Independence of paths at this level reflects context freeness of rewriting and suggests why they can be recognized efficiently. As suggested in Section 4.3.2, a derivation with independent paths can be divided into subcomputations with limited sharing of information.

We outlined the definition of a family of constrained grammatical formalisms, called Linear Context-Free Rewriting Systems. This family represents an attempt to generalize the properties shared by CFG's, HG's, TAG's, and MCTAG's. Like HG's, TAG's, and MCTAG's, members of LCFRS can manipulate structures more complex than terminal strings and use composition operations that are more complex that concatenation. We place certain restrictions on the composition operations of LCFRS's, restrictions that are shared by the composition operations of the constrained grammatical systems that we have considered. The operations must be linear and nonerasing, i.e., they can not duplicate or erase structure from their arguments. Notice that even though IG's and LFG's involve CFG-like productions,

they are (linguistically) fundamentally different from CFG's because the composition operations need not be linear. By sharing stacks (in IG's) or by using nonlinear equations over f-structures (in FUG's and LFG's), structures with unbounded dependencies between paths can be generated. LCFRS's share several properties possessed by the class of *mildly context-sensitive* formalisms discussed by Joshi (1983/85). The results described in this paper suggest a characterization of mild context-sensitivity in terms of generalized context-freeness.

Having defined LCFRS's, in Section 4.2 we established the semilinearity (and hence constant growth property) of the languages generated. In considering the recognition of these languages, we were forced to be more specific regarding the relationship between the structures derived by these formalisms and the substrings they span. We insisted that each structure dominates a bounded number of (not necessarily adjacent) substrings. The composition operations are mapped onto operations that use concatenation to define the substrings spanned by the resulting structures. We showed that any system defined in this way can be recognized in polynomial time. Members of LCFRS whose operations have this property can be translated into the ILFP notation (Rounds, 1985). However, in order to capture the properties of various grammatical systems under consideration, our notation is more restrictive that ILFP, which was designed as a general logical notation to characterize the complete class of languages that are recognizable in polynomial time. It is known that CFG's, HG's, and TAG's can be recognized in polynomial time since polynomial time algorithms exist in for each of these formalisms. A corollary of the result of Section 4.3 is that polynomial time recognition of MCTAG's is possible.

As discussed in Section 3, independent paths in tree sets, rather than the path complexity, may be crucial in characterizing semilinearity and polynomial time recognition. We would like to relax somewhat the constraint on the path complexity of formalisms in LCFRS. Formalisms such as the restricted indexed grammars (Gazdar, 1985) and members of the hierarchy of grammatical systems given by Weir (1987) have independent paths, but more complex path sets. Since these path sets are semilinear, the property of independent paths in their tree sets is sufficient to cause semilinearity of the languages generated by them. In addition, the restricted version of CG's (discussed in Section 6) generates tree sets with independent paths and we hope that it can be included in a more general definition of LCFRS's containing formalisms whose tree sets have path sets that are themselves LCFRL's (as in the case of the restricted indexed grammars, and the hierarchy defined by Weir).

LCFRS's have only been loosely defined in this paper; we have yet to provide a complete set of formal properties associated with members of this class. In this paper, our goal has been to use the notion of LCFRS's to classify grammatical systems on the basis of their strong generative capacity. In considering this aspect of a formalism, we hope to better understand the relationship between the structural descriptions generated by the grammars of a formalism, and the properties of semilinearity and polynomial recognizability.

References

Berwick, R., 1984. Strong generative capacity, weak generative capacity, and modern linguistic theories. *Comput. Ling.* 10:189–202.

Berwick, R. and Weinberg, A., 1984. *The Grammatical Basis of Linguistic Performance.* MIT Press, Cambridge, MA.

Bresnan, J. W.; Kaplan, R. M.; Peters, P. S.; and Zaenen, A., 1982. Cross-serial Dependencies in Dutch. *Ling. Inquiry* 13:613–635.

Chandra, A. K.; Kozen, D. C.; and Stockmeyer, L. J., 1981. Alternation. *J. ACM* 28:114–122.

Gazdar, G., 1985. *Applicability of Indexed Grammars to Natural Languages.* Technical Report CSLI-85-34, Center for Study of Language and Information.

Gazdar, G.; Klein, E.; Pullum, G. K.; and Sag, I. A., 1985. *Generalized Phrase Structure Grammars.* Blackwell Publishing, Oxford. Also published by Harvard University Press, Cambridge, MA.

Joshi, A. K., 1985. How Much Context-Sensitivity is Necessary for Characterizing Structural Descriptions — Tree Adjoining Grammars. In Dowty, D.; Karttunen, L.; and Zwicky, A. (editors), *Natural Language Processing — Theoretical, Computational and Psychological Perspective.* Cambridge University Press, New York, NY. Originally presented in 1983.

Joshi, A. K., 1987. An Introduction to Tree Adjoining Grammars. In Manaster-Ramer, A. (editor), *Mathematics of Language.* John Benjamins, Amsterdam.

Joshi, A. K.; Levy, L. S.; and Takahashi, M., 1975. Tree Adjunct Grammars. *J. Comput. Syst. Sci.* 10(1).

Parikh, R., 1966. On Context Free Languages. *J. ACM* 13:570–581.

Pollard, C., 1984. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language.* PhD thesis, Stanford University.

Rounds, W. C. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. To appear in *Comput. Ling.*

Rounds, W. C., 1969. Context-free Grammars on Trees. In *IEEE 10th Annual Symposium on Switching and Automata Theory.*

Steedman, M. J., 1985. Dependency and Coordination in the Grammar of Dutch and English. *Language* 61:523–568.

Steedman, M., 1986. Combinatory Grammars and Parasitic Gaps. *Natural Language and Linguistic Theory* (to appear).

Thatcher, J. W., 1973. Tree Automata: An informal survey. In Aho, A. V. (editor), *Currents in the Theory of Computing,* pages 143–172. Prentice Hall Inc., Englewood Cliffs, NJ.

Weir, D. J., 1987. *Context-Free Grammars to Tree Adjoining Grammars and Beyond.* Technical Report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia.