PROLEGOMENA TO A FORMAL THEORY OF DEPENDENCY GRAMMAR

Norman M. Fraser[1]

Abstract

Phrase structure grammar and dependency grammar are alternative systems for the generation of syntactic structures. A great deal of effort has gone into proving the mathematical properties of different classes of phrase structure grammar. In contrast, very little work has been done on the formal properties of dependency grammars and what has been established relates to only one of many possible classes of dependency grammar. In this paper we review some relevant facts about phrase structure grammar and compare what is known about dependency grammar. We suggest some ways of defining new classes of dependency grammar by modifying rule formats, by allowing multiple heads, and by revising the adjacency constraint.

1. Introduction

Phrase structure grammar (PSG) and dependency grammar (DG) are alternative systems for representing syntactic structure. Their traditions are quite different. PSG has its origin in Bloomfield's (1933) immediate constituent analysis. It was formalized as a generative system by Chomsky (1956) who identified a number of varieties of PSG. One of these, context free PSG (CFPSG), proved to be of particular interest to computer scientists, who studied it as a formal system for the representation of programming languages. A great deal is now known about CFPSG and many results have been proven mathematically. (The more important of these are summarised in Aho and Ullman 1972). The role of PSGs as the base component of transformational grammars (Chomsky 1957, 1965) enabled results from formal language theory to feed directly into the study of natural languages. With the introduction of such extensions as the X' mechanism and the use of featural categories, PSG was elevated to a position of prime importance in monostratal theories of natural language (eg. Gazdar 1982; Gazdar *et al.* 1985), whilst retaining its existing role in transformational theories.

The story of DG is rather longer than that of PSG, dating back at least as far

[1]Present address: Social & Computer Sciences Research Group, University of Surrey, Guildford, Surrey, England, GU2 5XH.

as the medieval speculative grammarians and perhaps much further; Covington (p.c.) identifies the Arabic grammarians as a possible source. The works of Jespersen (1924), Bloomfield (1933), Hjelmslev (1939), Harris (1946) and Hockett (1958) all recognised the existence of dependencies between heads and modifiers, and could thus, in a loose sense, be termed 'dependency grammars'. However, it was Tesnière (1953; 1959) who articulated the first abstract theory of DG. Already we see a significant difference in the development of PSG and DG. The formal apparatus of PSG was developed first and then it was applied to the analysis of natural languages. The notions of DG have been adduced in the study of natural language for centuries but they were not formalized until comparatively recently. Although PSG and DG were first given formal expression at roughly the same time, and in spite of the fact that (unformalized) DG boasts a much longer history, PSG has had pride of place in modern linguistic theory while DG has lain largely unexplored.

There are many possible explanations for this disparity. Some of these may concern the personalities and politics invested in the study of each framework. The attractiveness of the larger frameworks within which they were being explored (eg. transformational grammar) was probably a contributing factor. A simple desire to break with the past should not be ignored. The extent to which any or all of these contributed to the relative unpopularity of DG is not known. While it would be interesting from the point of view of social history, the answer could have little relevance to modern linguistics. However, there is one possible contributory factor which deserves further attention since it may bear directly upon current problems: in comparison with PSG, the formal theory of DG is extremely under-developed. In the absence of any other persuasive evidence, the better understood system will always be selected. However, the lack of hard facts about DG leaves open the question of whether either formal system is inherently 'better' than the other at capturing the facts of natural language.

This paper presents a critical review of what is thought to be known about the formal properties of DG; it identifies a number of open questions concerning DG, and sketches some possible solutions.

We begin with a brief overview of the main tenets of the formal theory of PSG. Use of arcane symbols will be kept to a minimum. For a simple introduction to formal language theory see Rayward-Smith 1983.

2. Phrase structure grammar

2.1 The form of PSGs

A PSG is a 4-tuple of the form:

$$G = \{N, \Sigma, P, S\}$$

where

1. N is a finite set of *nonterminal symbols*. (These are the phrase names or syntactic categories).
2. Σ is a finite set of *terminal symbols*, ie. the alphabet. Σ is disjoint from N.
3. P is a finite subset of

$$(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

   ie. the elements of P will have the form (α,β), where α consists of a nonterminal which may be preceded or followed by any number of terminals or nonterminals. β consists of any number of terminals or non-terminals, including none (the empty set e). The elements of P are usually written $\alpha \rightarrow \beta$ and are called *productions* or *rules*.
4. S is a symbol in N and is called the *start symbol*.

A production rule is a mapping from one set of nonterminal and terminal symbols corresponding to a particular string, to another set of nonterminal and terminal symbols corresponding to the same string.

For example, a grammar which generates three ones followed by arbitrarily many pairs of alternating zeros and ones (eg. 1110101, 111, 1110101010101010l) would be defined as follows: $\{\{S,A\},\{0,1\},\{(S,111A),(A,01A),(A,e)\},S\}$. The productions are easier to read if they are presented in $\alpha \rightarrow \beta$ format, as in (1).

(1) a. $S \rightarrow 111A$
    b. $A \rightarrow 01A$
    c. $A \rightarrow e$

Notice that all of the information present in the 4-tuple definition of the grammar is recoverable from the productions in (1).

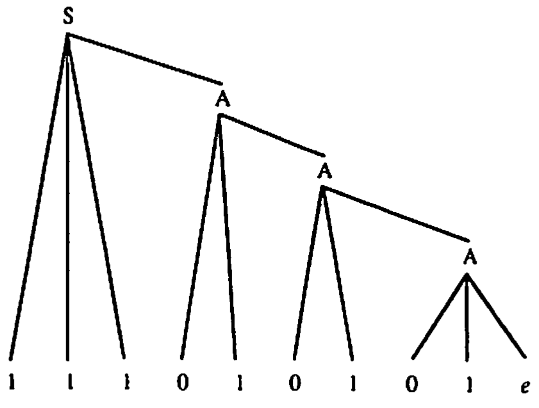
A number of observations can be made about the grammar in (1). All of the rules have exactly one nonterminal to the left of the arrow; this is interesting for reasons which will emerge in the next section but it is not necessary given our definition of PSG. The nonterminal A appears on both sides of the arrow in rule (1b). This makes the rule *recursive*, that is, the production rule contains an embedded reference to itself. It is recursion which, in general, allows a finite grammar to generate an infinite language, and which in this specific case allows the grammar to generate infinitely many 01 pairs. It is usually the case that grammars are required to generate arbitrarily long, rather than infinitely long, strings so most recursive rules require an 'escape hatch' to prevent them from looping indefinitely. In this grammar it is the null rewrite rule (1c) which terminates the recursion. Recursion is not confined to single rules having the same nonterminal on each side; it can equally be brought about by the interaction of several rules. For example, the following rules introduce recursion: $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$. Although the recursion is distributed across three rules, the effect is the same as if it had been confined to a single rule.

We can show the derivation of a string, eg. 111010101, as follows:

$$
\begin{aligned}
(2)\quad S &\Rightarrow 111A \\
&\Rightarrow 11101A \\
&\Rightarrow 1110101A \\
&\Rightarrow 111010101A \\
&\Rightarrow 111010101
\end{aligned}
$$

Here, we have rewritten one nonterminal on each line of the derivation. The same information can be portrayed more helpfully in a tree, whose root is the start symbol, whose leaves are terminals, and whose branches divide at nonterminal-labelled nodes.

(3)



PSGs are sometimes known as *Chomsky grammars* in recognition of the part played in their development by Noam Chomsky. Chomsky's introduction of PSGs in the late 1950s and early 1960s led to the most fruitful period of research in formal language theory to date. A particularly useful contribution was his language taxonomy which has come to be known as the *Chomsky hierarchy*.

2.2 The Chomsky hierarchy

The Chomsky hierarchy of languages (Chomsky 1956, 1959) is defined by reference to the grammars which generate the languages. The more restrictions that are placed on the form of a grammar, the more restricted will be the range of languages which can be generated by that grammar.

Type 0: Unrestricted languages

Suppose no restrictions are placed on the form of the productions in a grammar, beyond the restriction implicit in the basic definition of PSG, namely

$$\alpha \rightarrow \beta$$

where α and β are arbitrary strings of (terminal or nonterminal) symbols and $\alpha \neq e$. Grammars of this form are obviously known as the *phrase structure* grammars; they are also known as *type 0, semi-Thue,* and *unrestricted* grammars. The languages generated by such grammars form a class which are similarly designated. They are also known as the *recursively enumerable sets.*

Type 1: Context-sensitive languages

Suppose a simple restriction is placed on the form of rules, namely the stipulation that in rules of the form

$$\alpha \rightarrow \beta$$

β must be at least as long as α. This has the effect of eliminating from the grammar rules like (4) but allowing rules like (5).

(4) a. aBC → aD
    b. AB → C

(5) a. aBc → abc
    b. AC → BC

A grammar of this kind is known as a *type 1* or *context-sensitive* grammar. The class of languages generated by rules of this form are likewise called type 1 or context-sensitive languages. It is very hard to conceive of a language which is not context sensitive. The only languages which have been proven to be non-context sensitive (by complex diagonalization proofs) are not easily describable (Hopcroft and Ullman 1979:224).

Type 2: Context free languages

Suppose a further restriction is placed on the form of production rules, namely the stipulation that in rules of the form:

$$\alpha \rightarrow \beta$$

α must be exactly one nonterminal symbol, while β can be any string of (terminal and nonterminal) symbols. Grammars of this form and their related languages are termed *type 2* or *context-free.* Neither (4) nor (5) are context-free; (6) is context-free.

(6) a. $A \rightarrow BC$
    b. $A \rightarrow De$
    c. $A \rightarrow f$

Notice that these rules can not be interpreted as saying 'in the context of some preceding symbol and/or some following symbol rewrite symbol X as Y.' Instead, they must be taken to mean 'in any and every context, rewrite X as Y.'

Type 3: Regular languages

Suppose a final restriction on the form of a production is introduced, namely the stipulation that in a rule of the form

$$\alpha \rightarrow \beta$$

α must be exactly one nonterminal symbol, while β can consist of either a single terminal or a terminal followed by a nonterminal, as in (7):

(7) a. $A \rightarrow a$
    b. $A \rightarrow aB$

If all productions take this form, the grammar is said to be *right-linear*. If, instead, β consists of either a single terminal or a nonterminal followed by a terminal, as in (8), the grammar is said to be *left-linear*.

(8) a. $A \rightarrow a$
    b. $A \rightarrow Ba$

For every right-linear grammar, there exists a left-linear grammar which generates the same string set and *vice versa*. Right- and left-linear grammars are known as *type 3* or *regular* grammars. The single set of languages generated by both kinds of regular grammar is also called the set of type 3 or regular languages.

Proper containment

A set of languages L_1 is *properly contained* in another set L_2 if every language in L_1 is a member of L_2, ie. L_1 is a subset of L_2. Leaving aside the empty string which causes a minor problem, any language of type n is properly contained in the language of type $n - 1$, where $n \in \{1,2,3\}$. In other words, the regular languages are properly contained in the context-free languages, the context-free languages not containing the empty string are properly contained in the context-sensitive languages, and the context-sensitive languages are properly contained in the unrestricted languages. Formal proofs of the *Hierarchy Theorem* (as the above generalisation is called) can be found in Aho and Ullman 1972: Chapter 2 and Hopcroft and Ullman 1979: Chapter 9.

3. Dependency grammar

So far we have considered a variety of grammar which groups objects (terminal symbols or phrases) into larger objects (phrases). In other words it encodes the *constituent structure* of sentences. However, there is an alternative view of sentence structure which takes the notion of directed dependence as central. The grammar is made to encode not the vertical relations between items and categories, but rather the horizontal relations which hold between terminal symbols. The insight here is that the presence of some item demands or allows the co-presence of some other item. If it is possible to identify some sort of logical dependency holding between symbols then it will be possible to describe a *dependency structure* for sentences which is orthogonal to the constituent structure of those sentences.

For example, consider the language consisting of the integers between -9 and 9 (excluding 0 for simplicity). The vocabulary of this language is {*1,2,3,4,5,6,7,8,9*,+,-}. This vocabulary can be divided into the number symbols {*1,2,3,4,5,6,7,8,9*} and the sign symbols {+,-}. A well-formed string of this language consists of either a number symbol or a number symbol preceded by a sign symbol. So, *5*, -*5*, and +*5* are all well-formed strings of the language. Notice that a sign symbol can only appear in a string containing a number symbol but a number symbol can appear without a sign symbol. This asymmetry can be captured in a DG.

There is some terminological variation in the literature. Elements which depend on others are called *dependents* or *modifiers*; elements on which others depend are called *heads*, *governors*, *controllers* or *regents*. We shall use the terms *head* and *dependent*.

3.1 The form of a dependency grammar

Considerably less formal work has been done on DG than on PSG. Gaifman has developed the nearest thing to a 'standard' notation for DG rules (Gaifman 1965). The rule formats are shown in (9).

(9) a. $X_i(X_{j1},X_{j2},\ldots,*,\ldots,X_{jn})$
b. $X_i(*)$
c. $*(X_i)$

In rules of this type, the element outside the brackets is the head of the construction and the asterisk denotes that element. In rule type (9a), X_i is the head and X_{j1} to X_{jn} are its dependents, where $n > 0$. The asterisk marks the position of the head relative to its dependents in the string. Rule type (9b) notates the case where X_i can occur without dependents. Rule type (9c) notates the case where X_i can occur without depending on any other element.

Using these conventions we can write a grammar to generate the single digit integer language we defined above (where N is one of the digits and S is one of the signs):

(10) $*(N)$
$N(*)$
$N(S,*)$

So far we have only mentioned the local relations which hold between heads and dependents. However, it is necessary to make explicit a number of general constraints on the form a grammatical sentence can take. The following constraints are usually assumed.

(11) a. One and only one element is independent;
b. all other elements depend on some element;
c. no element depends directly on more than one other; and
d. if A depends directly on B and some element C intervenes between them (in linear order of string), then C depends directly on A or on B or on some other intervening element. (Robinson 1970:260)

We shall consider each of these constraints in turn.

One independent element

In PSG the start symbol is a distinguished non-terminal symbol which rewrites, ultimately, as a legitimate string of the language. This means that the grammar can refer directly to sentences. For example, if the start symbol is *S*, a sentence is a phrase of type *S*. In DG there is nothing which directly identifies a sentence. The grammar encodes permitted relationships rather than permitted phrases. However, without referring directly to sentence-type objects, a DG can generate sentences. In a dependency structure every word must depend on some other word with the exception of one word - the sentence *root* - which is free. In a DG, the root symbol (terminal) rather than the sentence symbol (non-terminal) is marked as the start symbol.

Connected dependency structures

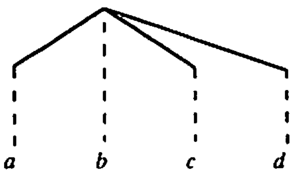
In PSG all of the symbols in a sentence must form part of the same phrase structure; likewise, in DG all of the symbols in a sentence must form part of the same connected dependency structure. A word must be either the root or a *subordinate* of the root.

Only one head

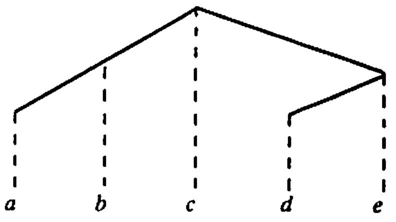
Each symbol (except the root) must depend on exactly one other symbol. This is rather like the constraint which is implicit in our characterisation of PSG, that a symbol or phrase may belong to only one category at a time; phrases must not overlap.

Heads and dependents are adjacent

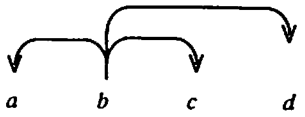
Let us consider the dependency structures we have defined so far. Dependency relations hold between the symbols in a string; one of these symbols depends on nothing at all, while all the others depend on exactly one other symbol. There are at least two conventions for presenting such dependency structures diagrammatically. In the first, exemplified in (12), dependency is represented by the relative vertical position of nodes corresponding to symbols in a tree; if a line connects a lower node to a higher node then the symbol corresponding to the lower node depends on the one corresponding to the higher node. We shall call this kind of diagram a *stemma* (following Tesnière 1959). In (12), *a, c* and *d* depend directly on *b*, which is the sentence root.

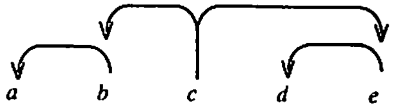
(12)


A more complex example is given in (13). In this case, while *b* and *e* depend directly on the sentence root *c*, the relationships between *a* and *d* and the root are indirect. Notice also that *e* depends directly on *c* although they are separated by *d* which does not depend directly on *c*.

(13)


An alternative diagrammatic convention moves away from using tree-like objects which are confusingly suggestive of PSG structures, and instead represents dependency relations by means of directed arcs. We shall adopt the convention of directing arcs from heads to dependents, although (unfortunately) there is no generally accepted labelling convention and it is not unusual to find examples in the literature of arcs being oppositely directed. We shall, for the time being, refer to diagrams of this kind as *arc diagrams*. Diagrams (14) and (15) are equivalent to (12) and (13) respectively.

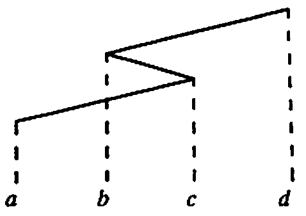
(14)


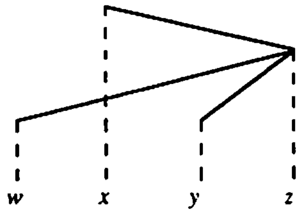
(15)


A rule of the form *A*(*B*,*) places no constraints on where B may be located, so long as it precedes *A* in the string. In principle, *A* and *B* could be separated by arbitrarily many other symbols which depend on neither *A* nor *B*. However, most work to date has concentrated on a class of dependency grammars which place an additional constraint on the location of dependents, namely that presented in (11d) above. This constraint is reproduced here for convenience.

(11) d. If A depends directly on B and some element C intervenes between them (in linear order of string), then C depends directly on A or on B or on some other intervening element. (Robinson 1970:260)

This constraint can be seen to be satisfied in the above dependency structures. The constraint is violated in the dependency structures shown in (16) and (17) so these must be declared ungrammatical according to the version of DG under consideration here.

(16)


(17)


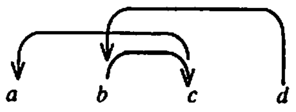
In (16), *a* violates the constraint. *a* is separated from its head *c* by *b* which depends on neither *a* nor *c*, neither does it depend on any element intervening between *a* and *c*. In a stemma, the dotted line which connects a symbol with its node is called its *projection*. It will be noted that in examples (12) and (13), links and projections do not intersect. Such diagrams and their corresponding syntactic structures are said to be *projective* (Lecerf and Ihm 1963). In (16) a link and a projection are seen to intersect at precisely the point where we detected ill-formedness. The same is also true in (17) . Here *w* is separated from its head *z* by *x* and *y*. *y* is a dependent of *z* so the positional constraint is not violated here. Neither are there any intersections in the stemma involving the projection of *y* and the link between *w* and its head *z*. However, the positional constraint forbids the intervention of the sentence root *x* between *w* and its head. Again, exactly at the trouble spot, an intersection is found in the stemma. Diagrams like (16) and (17), and the corresponding syntactic structures are said to be *non-projective*.

The vocabulary of projectivity is rooted in the imagery of stemmas. A less diagram-dependent terminology is preferable. We shall say that dependency relations can only be established between *adjacent* items, where adjacency is defined as follows:

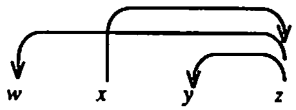
(18) A is adjacent to B iff
A is next to B or
A is separated from B by C and C depends on either A or B, or on a subordinate of A or B.

Notice that arcs never cross in arc diagrams of structures which satisfy the adjacency condition, whereas arcs do cross where the structures violate the adjacency condition. (19) and (20) show the arc diagram equivalents of (16) and (17) .

(19)



(20)



Given that the two styles of diagram seem to be equivalent, there appears to be little to choose between them. However, as we shall discover below when we try to extend the dependency formalism, the two styles make different predictions and the tree diagram will be found wanting. In anticipation of this we shall adopt the arc diagram henceforth.

A simple DG could be expressed as follows:

$$D = \{\Sigma, \mathcal{D}\}$$

where Σ is a set of symbols and $\mathcal{D}$ is a set of dependency rules. Given this definition it is not possible to make generalisations over sets of symbols. There would have to be two sets of almost identical rules for any two symbols with the same distribution. To overcome this, most practical dependency grammars are expressed as 4-tuples having the form:

$$D = \{\Sigma, C, \mathcal{A}, \mathcal{D}\}$$

where Σ is a set of symbols, C is a set of category labels, $\mathcal{A}$ is an assignment function which assigns symbols to categories, and $\mathcal{D}$ is a set of dependency rules. Given this definition it is possible for rules to express dependency relations holding between symbols and symbols, between symbols and categories of symbol, and between categories and other categories. We used this shorthand notation in the grammar of positive and negative digits (10), in which N stood for the symbols 1-9 and S stood for the signs + and -. Some writers define DGs as 5-tuples of the form:

$$D = \{\Sigma, C, \mathcal{A}, \mathcal{D}, S\}$$

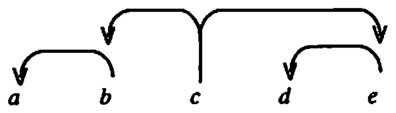
where S is a set of distinguished start symbols. Whether the start symbols are identified in $\mathfrak{D}$ or in S is purely a matter of taste; there are no implications for the generative capacities of the grammars the formalisms notate.

Notice that we have not said anything in these formalisations about general restrictions such as the single head and adjacency constraints. These must be stated elsewhere. Formal definitions of these constraints can be found in Gaifman 1965: 306.

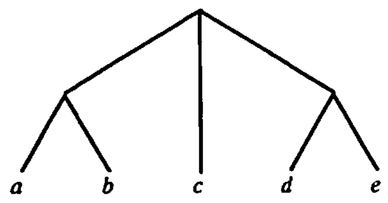
3.2 (Some) dependency grammars generate context free languages

In comparing different grammars two main factors need to be taken into account. Given any two grammars, (i) do they generate the same set of strings and (ii) do the structures assigned to strings by each grammar correspond systematically to structures in the other grammar? If two grammars generate the same string sets they are said to be *weakly equivalent*. If the string sets are generated with systematically corresponding structures they are said to be *strongly equivalent*. It is useful to be able to compare specific grammars but we are more concerned here to compare general frameworks, namely PSG and DG. If, for every grammar in one framework there exists a weakly equivalent grammar in another framework, then the two are said to be *weakly equipotent*. If, for every grammar in one framework there exists a strongly equivalent grammar in the other framework, the two are said to be *strongly equipotent* (for this usage see Hays 1964:519).

Consider the following structure:

(21) 

This dependency structure could be simply converted into a constituent structure by defining a constituent to consist of a word plus all its subordinates. Thus (21) would be analysed as [[*a b*] *c* [*d e*]], i.e.:

(22) 

It is hard to conceive of any grammatical dependency structure which would not have a corresponding phrase structure. This would suggest that every string which can be generated by a DG can also be generated by a PSG. This is half way to saying that DG and PSG are weakly equipotent. Before we consider the other half of the argument - whether every string generated by a PSG can also be generated by a DG - let us look briefly at the amount of information encapsulated in each representation. In (22) phrase structure is shown explicitly; in (21) phrase structure can be derived from the dependency relations so the representations can be said to encode the same amount of phrase structure information, although non-terminals are missing from the dependency structure. In (21), the relations between the symbols are directed; however, in (22) that information is lost and can not be recovered. The most that can be said about *a* and *b* is that some dependency relation holds between them. The direction of that dependency can not be established. Thus, we can immediately rule out the possibility of DG and simple PSG being strongly equipotent. The structure shown in (22) conflates the differences between four possible dependency structures. This is because it is not clear whether *a* depends on *b* or *vice versa*, or whether *d* depends on *e* or *vice versa*.

Although DG and PSG are not strongly equipotent, Gaifman has proved that DG and one type of PSG are weakly equipotent (Gaifman 1965). His proof is indirect and rests on a proof that DG and categorial grammar are weakly equipotent. Given that categorial grammar and CFPSG are known to be weakly equipotent (Bar-Hillel *et al.* 1960) we may conclude that DG and CFPSG are weakly equipotent.

Gaifman's proof is complex and confusing. As well as showing the weak equipotence of DG and CFPSG, he claims to demonstrate the strong equivalence of DGs and a subset of CFPSGs. There are two problems with his exposition. First, his definition of the relevant subclass of CFPSGs is opaque. Second, he defines strong equivalence solely in terms of phrase structure. Thus, according to his reasoning, since it is possible to recover a phrase structure from a dependency structure, there must be strong equivalence. He fails to acknowledge the problems in recovering a dependency structure from a phrase structure. This lack of clarity and his idiosyncratic definition of strong equivalence have led to a number of subsequent misunderstandings. For example, Lyons expresses uncertainty: "dependency grammars...are weakly, and perhaps strongly, equivalent to phrase structure grammars" (1970: 88). Robinson goes further, seeming to claim strong equivalence: "For every structure-free [i.e. context free] DG there is a strongly equivalent structure-free PSG (Gaifman 1965), and for every structure-free PSG there is a systematically corresponding structure-free DG" (1970:263). Hudson certainly interprets her as making a claim for strong equivalence: "Robinson has shown that dependency structures and constituency structures are formally equivalent, in the sense that one can be converted in a mechanical way into the other" (1976: 199). Matthews, on the other hand, argues against strong equivalence (1981: 84ff).

The terms of this debate were significantly affected by the introduction of the mechanisms of *X' grammar* (eg. Jackendoff 1977). X' grammars are CFPSGs in which the head of each phrase is marked explicitly. The general form of an X' rule is:

$$X^{n+1} \rightarrow \ldots X^{n} \ldots$$

In principle, this should make it possible to establish the strong equivalence of PSGs and DGs. However, there remain a number of uncertainties about the formal properties of X' grammars (Pullum 1985) so the debate is by no means closed.

4. Extending the dependency formalism

We have seen how the 'standard' version of DG has been shown to be weakly equivalent to CFPSG and possibly strongly equivalent to X' grammar. I would suggest that these formal results have had a detrimental effect on research in DG since it has seemed legitimate to dismiss DG as a 'notational variant' of a more familiar system, CFPSG. However, I would like to argue that, in fact, these results only relate to one of many possible definitions of DG and that alternative definitions may generate significantly different string sets and trees. We shall target three DG constraints for further attention: the form of dependency rules, the single head constraint, and the adjacency constraint.

4.1 The form of dependency rules

We have seen how the imposition of increasingly stringent constraints on the form of PSG rules leads to the generation of increasingly restricted languages. In principle, there is no reason why a similar exercise should not be carried out with DG rules, leading to the definition of a hierarchy of languages which is similar or identical to the Chomsky hierarchy. Although this is a very simple suggestion, it has not - to my knowledge - been discussed in the literature. We shall consider some ways in which the form of DG rules can be modified with possible implications for the generative capacities of the resulting grammars.

In Gaifman's DG notation, rules may have the form shown in (9) (reproduced here for convenience.

(9) a. $X_i(X_{j1},X_{j2},\ldots,*,\ldots,X_{jn})$
b. $X_i(*)$
c. $*(X_i)$

A maximally constrained DG would restrict rules to the following forms:

(23) a. $X_i(X_j,*)$
b. $X_i(*)$
c. $*(X_i)$

These rules allow each symbol to have at most one dependent symbol. All dependencies point in the same direction (right to left). Alternatively, all dependencies could point in the opposite direction:

(24) a. $X_i(*,X_j)$
b. $X_i(*)$
c. $*(X_i)$

Clearly, the set (or sets) of languages generated by grammars of this kind is (or are) properly contained in the set of languages generated by grammars conforming to (9) since rules of type (23) and (24) are well-formed rules of type (9). We may hypothesize that (23) and (24) generate the type 3, or regular, languages, although we shall not examine any proofs here. There is a clear similarity between left-linear type 3 PSGs and DGs expressed in the rule format of (23) and between right linear type 3 PSGs and DGs expressed in the rule format of (24).

Consider the language $\{a^nb^n | n \geq 0\}$. It is well known that CFPSGs can generate this language but regular grammars can not. Standard form DGs (9) can also generate this language:

(25) a. *(a)
b. b(*)
c. a(*,b)
d. a(*,a,b)

However, DGs of the form found in (23) and (24) can not generate this language since there is no way of ensuring that *a*s and *b*s match (this is the purpose of rule (25d).

It is worth pointing out that CFPSGs can only generate the language a^nb^n by means of centre-embedding but Gazdar has argued (1988) that natural language constructions of the form a^nb^n nest at the right or left periphery. The grammar in (25) nests at the left periphery and the language could equally easily be generated by a DG nesting at the right periphery.

Having considered a restricted version of Gaifman's rule formats, we now examine how they may be extended. One way would be to introduce an element of context-sensitivity. This could be accomplished by allowing all of the rule formats given in (9) but additionally allowing arbitrarily many symbols to appear outside the brackets. If one of the symbols outside the brackets is distinguished as head of the symbols inside the brackets (for example, by enclosure within square brackets), then the other symbols outside the brackets can serve to restrict the contexts in which the rule may apply. For example:

(26) ab[e]h(c,d,*,f,g)

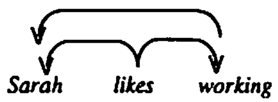
This rule will serve to rewrite an *e* just in those contexts where it is preceded by *ab* and followed by *h*. It is not yet clear whether this augmentation has any effect on the generative capacity of the formalism. It is worth noting that this kind of rule is additive, i.e. the symbols inside the round brackets are added to the symbol inside square brackets preserving the relative ordering described in the rule. The other symbols remain unchanged, serving only to describe context. A rewriting step will never decrease the number of symbols in the string. PSG rules are string replacement rules, i.e. what appears to the right of a rewrite arrow replaces everything that appears to its left. It is possible with type 0 PSGs that the number of symbols in a string will decrease after a rewrite operation. It is difficult to conceive of any set of rule formats which could reasonably be termed a DG which could ever erase symbols. If this is correct, then it could be argued that DG is a more restrictive framework and so is a more attractive option for the description of natural language than PSG.

An interesting set of unexplored questions relates to whether or not it is possible to develop a *normal form* for DGs of a particular class. For example, is it possible to convert an arbitrary *bidirectional* DG (in which dependencies point in both directions as in (9)) into a *unidirectional* DG (in which all dependencies point in the same direction)?

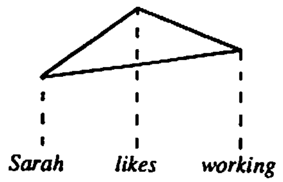
4.2 The single head constraint

From rule formats we turn to general constraints on the structure of well-formed strings. In particular we shall reconsider the stipulation (11c) that no element depends on more than one other element in a sentence. Alternative versions of DG have been advocated for the analysis of natural language in which words may have multiple heads. This position can be found in Hudson's Word Grammar (1984; 1990) and in Starosta's Lexicase (1988). They argue that multiple heads are required if the facts of natural language are to be fully described. For example, Hudson argues that in sentences like (27) *Sarah* is subject of both *likes* and *working*.

(27)



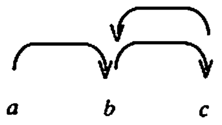
Earlier we suggested that stemmas and arc diagrams are not exactly equivalent; notice that arcs do not intersect in this diagram whereas projections do intersect in the corresponding stemma:



(28)

Hudson and Starosta are both very careful to restrict the circumstances under which multiple headedness may occur. They are wise to do so since there are many open questions about the real effects of introducing multiple headedness into the formalism. It certainly rules out the possibility of the resulting family of DGs being strongly equivalent to any PSG, even those of the X' variety. It is not clear what effect, if any, multiple headedness might have on the weak generative capacity of the system. There are further implications for the form which rules may take and the ways in which those rules should be interpreted.

Allowing multiple heads raises the possibility of generating structures including *interdependent* symbols. For example, in (29) *c* depends on *b* and *b* depends on *c*. No constraints have been relaxed other than the single head constraint.



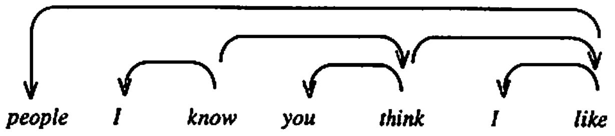
(29)

The formal implications of interdependence are not properly understood but any formal treatment of a system allowing multiple heads could not remain agnostic on the subject.

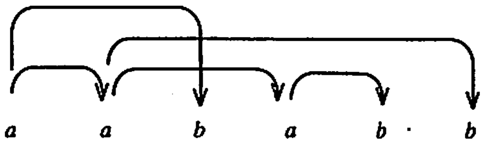
4.3 The adjacency constraint

It is widely agreed that what is required to account for the facts of natural language is a formalism with generative capacity between that of CFPSG and context sensitive PSG, somewhere around the capacity of the *indexed* languages (Gazdar 1988), also known as *mildly context sensitive* languages (Joshi 1985). In order to generate languages of this capacity and also to permit the construction of particular, semantically motivated structures, some DG theorists have proposed that the adjacency constraint be either abandoned or modified. Amongst those advocating the abandonment of the constraint are Pericliev and Ilarionov (1986) and Sgall, Hajicova and Panevova (1986). Those advocating modified versions of the adjacency constraint include Hudson (1984; 1990). We shall consider each of these approaches in turn.

An apparent advantage of abandoning the adjacency constraint is that it should be possible to generate structures like that in (30), in which *people* is separated from its head *like* by a number of elements which are dependent on neither, including *know* which is not dependent on anything at all.

(30)


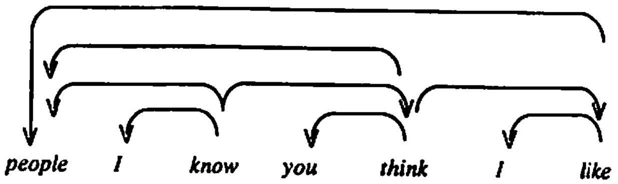
Indeed, for an analysis of natural language it is desirable to be able to generate just such a structure for this sentence. However, there are compelling reasons for believing that wholesale abandonment of the adjacency constraint is not the best way of legitimizing this kind of proscribed structure. Far from extending the power of a DG, abandonment of the adjacency constraint actually reduces the expressive power of the formalism. For example, it is no longer possible to generate the language a^nb^n. In a framework devoid of an adjacency constraint, the previously adequate rules in (25) would generate all of the strings in a^nb^n, plus many strings not in the desired language, eg.:

(31)


The alternative approach is to re-define the adjacency constraint so that it allows desired structures and blocks unwanted structures. Hudson offers the following re-formulation:

(32) D is adjacent to H provided that every word between D and H is a subordinate either of H, or of a mutual head of D and H. (1990:§6.4).

Hudson's definition of 'subordinate' is unusual in that every word counts as a subordinate of itself. This revised version of the adjacency constraint serves to prohibit structures like (31). It allows desired structures like (27), which violate the conventional version of the adjacency constraint. However, it also prohibits desired structures such as (30). Hudson's revised adjacency is part of a package which includes the use of semantically vacuous links (called *visitor* links) to mediate desired, non-standard dependency links in a 'hopping' analysis (Hudson, this volume). Hudson's version of (30) would look like this:

(33) 

Hudson may constrain his grammars by careful use of rules but serious problems remain with his formalism. For example, his version of the adjacency constraint would allow every possible string given a vocabulary, so long as it was maximally connected, i.e. every symbol was dependent on every other symbol (except one of the peripheral symbols which was root and so depended on no other symbol).

Both the abandonment of the adjacency constraint and its revision have serious implications for the generative capacities of DGs. However, not only is there an absence of formal results in this area but there is also an absence of analytic techniques. These too must be flagged as research questions.

5. Conclusion

We have reviewed the main points of what little formal work has been done in the study of DGs. We have seen that 'standard' DG is equipotent with CFPSG. We have described a more restricted version of DG and hypothesized that it is equipotent with the regular grammars. We have also outlined a less restricted version of DG but we have not investigated its generative capacity. Other open questions concern the possibility of converting DGs to a normal form, the effect of allowing multiple headedness and the effect of relaxing or discarding the adjacency constraint. If DG is to be taken seriously as a resource for linguistic theories, then these questions must be addressed.

This work could be interpreted simply as an attempt to make DG's formal basis as respectable as that of PSG. That is certainly an issue of concern, but there is another, more important reason for following this line of inquiry. While the landscape of DG remains unexplored, it will never be known what previously unimagined classes of language lie waiting to be discovered.

References

Aho, A.V. and J. D. Ullman (1972) *The Theory of Parsing, Translation, and Compiling, Volume 1: Parsing*. Englewood Cliffs, N.J.: Prentice Hall.

Bar-Hillel, Y., H. Gaifman and E. Shamir (1960) On categorial and phrase structure grammars. *Bulletin of the Research Council of Israel* 9, Section F, 1-16. Also in Y. Bar-Hillel (ed) *Language and Information*. Reading, Mass.: Addison-Wesley.

Bloomfield, L. (1933) *Language*. New York: Holt, Rinehart & Winston.

Chomsky, N. (1956) Three models for the description of language. *IEEE Transactions on Information Theory* 2, 113-124.

Chomsky, N. (1957) *Syntactic Structures*. The Hague: Mouton.

Chomsky, N. (1959) On certain formal properties of grammars. *Information and Control* 2, 137-167.

Chomsky, N. (1965) *Aspects of the Theory of Syntax*. Cambridge, Mass.: MIT Press.

Gaifman, H. (1965) Dependency systems and phrse-structure systems. *Information and Control* 8, 304-337.

Gazdar, G. (1982) Phrase structure grammar. In P. Jacobson and G.K. Pullum (eds) *The Nature of Syntactic Representation*. Dordrecht: D. Reidel, 131-186.

Gazdar, G. (1988) Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer (eds) *Natural Language and Linguistic Theories*. Dordrecht, Holland: D. Reidel.

Gazdar, G., E. Klein, G.K. Pullum and I.A. Sag (1985) Generalized Phrase Structure Grammar. Oxford: Blackwell.

Harris, Z.S. (1946) From morpheme to utterance. *Language* 22, 161-183.

Hays, D. (1964) Dependency theory: a formalism and some observations. *Language* 40, 511-525.

Hjelmslev, L. (1939) La notion de rection. *Acta Linguistica* 1, 10-23.

Hockett, C.F. (1958) *A Course in modern Linguistics*. New York: Macmillan.

Hopcroft, J.E. and J.D. Ullman (1979) *Introduction to Automata Theory, Languages, and Computation*. Reading, Mass.: Addison-Wesley.

Hudson, R.A. (1976) *Arguments for a Non-Transformational Grammar*. Chicago: University of Chicago Press.

Hudson, R.A. (1984) *Word Grammar*. Oxford: Blackwell.

Hudson, R.A. (1990) *A Word Grammar of English*. Oxford: Blackwell.

Jackendoff, R. (1977) *X' Syntax: a Study of Phrase Structure*. Cambridge, Mass.: MIT Press.

Jespersen, O. (1924) *The Philosophy of Grammar*. London: Allen and Unwin.

Joshi, A.K. (1985) How much context sensitivity is necessary for characterizing structural descriptions - tree adjoining grammars. In D. Dowty, L. Karttunen and A. Zwicky (eds) *Natural Language Processing - Theoretical, Computational and Psychological Perspectives*. Cambridge: Cambridge University Press.

Lecerf, Y. and P. Ihm (1963) *Eléments pour une grammaire générale des langues projectives*. Brussels: Presses Académiques Européennes.

Lyons, J. (1970) *Chomsky*. Glasgow: Fontana/Collins.

Matthews, P.H. (1981) *Syntax*. Cambridge: Cambridge University Press.

Pericliev, V. and I. Ilarionov (1986) Testing the projectivity hypothesis. *COLING '86*. 56-58.

Pullum, G.K. (1985) Assuming some version of the X-bar theory. SRC-85-01, Syntax Research Centre, Cowell College, University of California, Santa Cruz.

Rayward-Smith, V.J. (1983) *A First Course in Formal Language Theory*. Oxford: Blackwell.

Robinson, J. (1970) Dependency structures and transformational rules. *Language* 46, 259-285.

Sgall, P., E. Hajicova and J. Panevova (1986) *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. Prague: Academia.

Starosta, S. (1988) *The Case for Lexicase: an outline of lexicase grammatical theory*. London: Pinter.

Tesnière, L. (1953) *Esquisse d'une Syntaxe Structurale*. Paris: Klincksieck.

Tesnière, L. (1959) *Eléments de Syntaxe Structurale*. Paris: Klincksieck.