A Grammar of Projections

Steven Abney
University of Tübingen

A sequence of nodes in a syntax tree, each the head of the next, is usually called a *projection path*, or simply a *projection*. Projections in this sense are fairly basic phrase-structural entities, though hardly of central importance in most accounts. There are a few exceptions. For example, varieties of projection that I called *c-projection* and *s-projection* were significant in my earlier work on the grammar of function words, and the distinction was further developed by Grimshaw under the rubric *extended projections*. And in quite a different direction, Kayne proposed *g-projections* to account for constraints on long-distance dependencies. Pesetsky, inspired by Kayne's g-projections, proposed using *paths* as the basis for an account of locality; and his paths, in turn, are reminiscent of the feature-passing paths that figure prominently in intuitive accounts of attribute-value grammars.

If there is more to this train of precedent than just free association and coincidences of terminology, it suggests that an account of both constituent structure and long-distance dependencies might be based on a single notion of projection. I would like to propose such an approach in the present work.

As a point of departure, let me make another set of free associations, not obviously related to the first. Traditionally, there have been two broad approaches to phrase structure: immediate-constituent analysis and dependency analysis. American structuralists mostly assumed immediate-constituent analysis in preference to dependency analysis, and that prejudice was inherited by Chomsky and transformational grammar.

But as transformational grammar has evolved, it has assigned an ever more central role to dependency concepts. Whereas dominance is the primitive structural relation in immediate-constituent analysis, government is the fundamental structural relation in dependency grammar. At least since the point at which the Extended Standard Theory evolved into Government-Binding Theory (GB), government has come to play a central role in Chomsky's work. (Some related notions had a prominent place in the theory much earlier, including headship and command—the latter being definable as the transitive closure of government.) I think it is not far from the mark to say that GB reconstructs dependency structures as defined terms, atop an immediate-constituency substrate. And with the more recent introduction of 'bare phrase structure' [**?**], Chomsky's phrase structures have all but abandoned the immediate-constituent substrate, and become nearly pure, if somewhat idiosyncratic, dependency stemmas.

But let us return now to the relation between dependency structure and constituent structure, and the question of what it has to do with projections. There is something rather unsatisfying about the opportunistic mixture of constituent structure and dependency structure in GB. Constituent grammars (i.e.,

context-free grammars) and dependency grammars are both straightforward, clean descriptions of what we might call phrase structure—using the term now in a pretheoretic sense to refer to the base syntactic structure, excluding relations of movement and construal. To understand the consequences of mixing them, it would be good to find a characterization of how "pure" constituency and "pure" dependency relate to one another. I would like to suggest that projections provide the key to the relation, in a manner I will describe shortly.

1 Constituency and Dependency

A classic paper by Gaifman [**?**] would appear to have laid to rest the issue of the correspondence between constituent grammars and dependency grammars, by proving that dependency grammars (DG's) are a special case of context-free grammars (CFG's). Gaifman proves that dependency grammars are equivalent to a proper subset of CFG's, the *degree-1* CFG's (to be defined below). Under a weaker notion of correspondence, which Gaifman attributes to Hays [**?**], dependency grammars correspond to *finite-degree* CFG's, which represent a larger subset of CFG's, but a proper subset nonetheless.

However, Gaifman obtains his result based on a correspondence between CFG's and DG's that suppresses headedness in DG's. Since headedness is central to DG's, Gaifman's characterization eviscerates them, casting serious doubt on the validity of his comparison.

I would like to propose a basis for comparison that preserves the essential features of both CFG's and DG's. In particular, I propose relating CFG's and DG's via what I will call *headed context-free grammars* (HCFG's). If we pursue this approach, it turns out that CFG's and DG's represent equivalence classes of HCFG's, but, contra Gaifman, the equivalence classes are orthogonal. Neither DG's nor CFG's subsume the other.

2 Trees

2.1 Ordered Trees

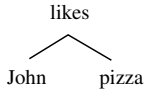
To examine Gaifman's theorem, we need to define more precisely what we mean by CFG's, DG's, HCFG's, and their languages. In these definitions, I do not follow Gaifman. Rather, I follow the systematization that has become standard since, modified to anticipate the needs of the second half of the paper, in which I discuss projections and locality. The content of Gaifman's result is readily expressed in the present system.

CFG's, DG's, and HCFG's generate *tree languages,* which I refer to simply as *languages,* inasmuch as my interest is strong rather than weak equivalences. The languages of the different types of grammars consist of different types of trees: constituent trees, dependency trees, and headed constituent trees, respectively. All three can be represented as special cases of *labelled trees* in which both

nodes and arcs are labelled. Labelled trees, in turn, are based on *ordered trees,* or simply "trees", which are the subject of this section.

The development here is somewhat nonstandard because I wish to define trees that generalize over both constituent trees and dependency trees. The essential difference between constituent trees and dependency trees is whether nodes are precedence-ordered with respect to descendants. In constituent trees, they are not, whereas in dependency trees, they are. For example, in the dependency tree

(1)



likes is precedence-ordered with respect to its children: it follows *John* and precedes *pizza*.

The approach I will take is to allow a node to appear among its own children, or rather, among its own *successors*. Each node has an ordered set of successors, and precedence relations are inherited from the orderings on successor sets. The children of a node x are the successors of x, excluding x itself.

Definition 1 *An* **ordered graph** *is a structure* (N, σ) *in which* N *is a finite set of* **nodes** *and* σ *is a function assigning to each node a sequence of nodes, its* **successors**.

In this paper, all graphs are understood to be ordered graphs.

This definition is not the usual one; usually an ordered graph is defined as a structure $(N, D, \prec)$ where $D \subseteq N \times N$ is the set of arcs and $\prec$ is the precedence relation. The arcs and precedence relation of a graph can be recovered from the successors, however, as follows.

Let $G = (N, \sigma)$ be an ordered graph. Then the set of **arcs** of G is $D = \{(x, y) | y \in \sigma(x)\}$;[1] that is, $D(x, y)$ iff y is a successor of x. **Ancestry** is the transitive closure of D and **descendance** is the inverse of ancestry. We write "$<$" for the ancestry relation, and $\leq$ is the reflexive closure of $<$, as usual. We define a node x to **precede** node y iff z, x', and y' exist such that $\sigma(z) = (\ldots, x', \ldots, y', \ldots)$ and $x' \leq_G x$, $y' \leq_G y$; we write "$\prec$" for the precedence relation in G. If necessary to avoid ambiguity, we will add a subscript to indicate which graph these sets and relations are to be understood with respect to, e.g., N_G, σ_G, D_G, $<_G$, $\prec_G$.

Note that a node y may appear multiple times in a given successor set σ. One consequence is that $<$ and $\prec$ are not sufficient to reconstruct $\sigma(\cdot)$ unambiguously. Another consequence is that $\prec$ cannot be guaranteed to have the properties we expect of a precedence relation. For example, if $\sigma(z) = (x, y, x)$

[1] Here and elsewhere, I treat sequences as if they were sets of nodes. By $y \in \sigma(x)$ is meant that y is a node in the sequence $\sigma(x)$, not that y is an element of $\sigma(x)$ viewed as a subset of $INI \times N$.

for some z and $x \neq y$, then we have simultaneously $x \prec y$ and $y \prec x$ (i.e., precedence fails to be antisymmetric), as well as $x \prec x$ (precedence fails to be antireflexive). Moreover, consider a graph with nodes $\{x, y, z\}$ such that $\sigma(x) = (x, y)$, $\sigma(z) = (y, z)$, and $\sigma(y) = \emptyset$. Then $x \prec y$ and $y \prec z$, but $x \not\prec z$ (precedence is not transitive). However, $\prec$ will be seen to be antireflexive, antisymmetric, and transitive in the special case of trees.

The **expansion** of a node x is the pair $(x, \sigma(x))$. If x is a self-successor, we write $x \to \alpha \cdot \beta$ for the expansion of x, marking the location of x with the dot. (In none of the examples of interest will x appear multiple times in $\sigma(x)$.) If x is not a self-successor, we write $x \to \sigma(x)$.

The set $\sigma(x) \backslash \{x\}$ constitutes the **children** of node x. Note that x may be a successor of itself, but not a child of itself. If x appears exactly once in $\sigma(x)$, then the **left children** of x are those successors preceding x in $\sigma(x)$, and the **right children** of x are those successors following x in $\sigma(x)$.

The transitive closure of childhood is **proper ancestry**, which we write $\leq\!\!\!\!/$ Proper ancestry differs from ancestry with respect to **self-successors**, that is, nodes that are their own successors. For self-successors x, we have $x < x$ but not $x \leq\!\!\!\!/\, x$. The reflexive closure of proper ancestry is identical to the reflexive closure of ancestry, and we write $\leq$ for both.

If y is a child of x then x is a **parent** of y. A node with no parent is a **root** and a node with no children is a **terminal node**. Nota bene that *root* and *terminal node* are defined in terms of child and not successor. If x is a root, there may exist a y such that $y < x$, namely if $x < x$. But if x is a root, there is no y such that $y \leq\!\!\!\!/\, x$. Similarly, if x is a terminal node, $\sigma(x)$ may be nonempty, but there is no y such that $x \leq\!\!\!\!/\, y$.

Note that there is in principle a distinction between self-successor terminal nodes and non-self-successor terminal nodes. This distinction is a nuisance rather than useful, so we will limit our attention to graphs in which all terminal nodes are self-successors. Making all terminal nodes self-successors probably seems counterintuitive, but will prove technically convenient. At the same time, we eliminate a second nuisance: successor sets containing duplicates. We define a **proper graph** to be a graph in which no successor set is empty, and no successor set contains duplicates, and we limit our attention henceforth to proper graphs.

Trees are proper graphs that satisfy additional conditions.

Definition 2 *A* **tree** *is a proper graph in which (1) there is exactly one root, (2) every node except the root has exactly one parent, and (3) the proper ancestry relation is acyclic.*

Note that the ancestry relation (as opposed to proper ancestry) may contain trivial cycles, because of the presence of self-successors ($x < x$ for some x).

Some basic properties of trees are given in the following theorem.

Theorem 1 *In a tree: (1) All pairs of nodes are ordered either by $\leq$ or by $\prec$. (2) Precedence is anti-reflexive, anti-symmetric, and transitive. (3) Precedence*

relations are preserved by ancestry. That is, if x precedes (follows, neither precedes nor follows) y and x is not a descendant of y, then x precedes (follows, neither precedes nor follows) every descendant of y. (4) Self-successors are precedence-related to all of their proper descendants, and non-self-successors are precedence-related to none of their proper descendants. (5) Terminal nodes are totally ordered by precedence.

Proof. First some preliminary comments. Since a tree t is single-rooted and acyclic and no node has multiple parents, it follows that there is a unique path from the root r to any given node in t. Let $p(x)$ be the path from r to x and $p(y)$ the path from r to y. Since there are no multiple parents, $p(x)$ and $p(y)$ must have a common prefix $(r = z_1, \ldots, z_n)$; z_n is the **greatest common ancestor**[2] (gca) of x and y. Let us also define $c_x(z)$ to be that child c of z such that $c \leq x$, if $z \neq x$, and $c_x(x) = x$. Clearly, $c_x(z)$ exists iff $z \leq x$, and it is unique if it exists. Recall that $x \prec y$ iff there is a u such that $\sigma(u) = (\ldots, x', \ldots, y', \ldots)$ with $x' \leq x$ and $y' \leq y$. Clearly, if u exists, it is a common ancestor of x and y. Since in a proper graph there are no duplicates in successor sets, $x' \neq y'$, so u must in fact be $\mathrm{gca}(x, y)$, with $x' = c_x(u)$ and $y' = c_y(u)$. That is, $x \prec y$ iff $\sigma(u) = (\ldots, c_x(u), \ldots, c_y(u), \ldots)$ for $u = \mathrm{gca}(x, y)$.

(1) Suppose x and y are not ordered by $\leq$. Let $u = \mathrm{gca}(x, y)$. Clearly $c_x(u) \neq u$, otherwise we would have $x = c_x(u) = u \leq y$, contrary to assumption; likewise $c_y(u) \neq u$. Also $c_x(u) \neq c_y(u)$, otherwise $c_x(u) = c_y(u)$ would be a common ancestor of x and y greater than $\mathrm{gca}(x, y)$. Therefore $c_x(u)$ and $c_y(u)$ are distinct members of $\sigma(u)$ for $u = \mathrm{gca}(x, y)$, thus $x \prec y$ or $y \prec x$.

(2) We have $x \prec y \prec x$ iff $\sigma(u) = (\ldots, c_x(u), \ldots, c_y(u), \ldots)$ and $\sigma(u) = (\ldots, c_y(u), \ldots, c_x(u), \ldots)$ with $u = \mathrm{gca}(x, y)$ and $c_x(u) \neq c_y(u)$. That is not possible, otherwise there would be at least one duplicate in $\sigma(u)$. It follows that precedence is anti-symmetric. Anti-reflexivity is the special case in which $x = y$.

Now consider x, y, z such that $x \prec y$ and $y \prec z$. Let $u = \mathrm{gca}(x, y)$; we have $\sigma(u) = (\ldots, c_x(u), \ldots, c_y(u), \ldots)$. If $\mathrm{gca}(y, z) = u$ then $\sigma(u) = (\ldots, c_x(u), \ldots, c_y(u), \ldots, c_z(u), \ldots)$ and $x \prec z$. If $u \leq \mathrm{gca}(y, z)$ then $\mathrm{gca}(x, z) = u$ and $c_y(u) = c_z(u)$, hence $x \prec z$. Finally, suppose $\mathrm{gca}(x, z) < u$. Let $u' = \mathrm{gca}(x, z)$; then $\mathrm{gca}(y, z) = \mathrm{gca}(x, z) = u'$ and $c_x(u') = c_y(u')$. Since $y \prec z$, $\sigma(u') = (\ldots, c_x(u') = c_y(u'), \ldots, c_z(u'), \ldots)$ and again $x \prec z$. This proves transitivity.

(3) Immediate from the fact that $c_z(u) = c_y(u)$ if $y \leq z$.

(4) If $x \leq y$, then $x = \mathrm{gca}(x, y)$, $c_x(x) = x$, $c_x(x) \neq c_y(x)$, and $c_y(x) \in \sigma(x)$. If $x < x$ then $c_x(x) = x \in \sigma(x)$ and x and y are precedence-related. If $x \not< x$ then $c_x(x) = x \notin \sigma(x)$, and x and y are not precedence-related.

(5) If x and y are distinct terminal nodes, they cannot be ancestry-related, so by (1) they must be precedence-related. ∎

We use self-succession to place nodes in precedence-order among their children, and hence, via the inheritence of precedence, within the terminal string or *yield* of the tree.

[2] "Greatest" because $z' < z$ for all other common ancestors z'.

Definition 3 *The* **yield** *of a tree* (N, σ) *consists of the self-successors in* N*.*

Obviously all terminal nodes belong to the yield, since all terminal nodes are self-successors. The yield is also the largest (totally) precedence-ordered set containing all terminal nodes. Since non-self-successors are not precedence-ordered with respect to their descendants, including in particular their terminal descendants, adding any non-self-successor to the yield would create a set that is not totally precedence-ordered.

Constituent tree and *dependency tree* are easily defined in terms of *yield*:

Definition 4 *An* **(unlabelled) constituent tree** *is a tree in which only terminal nodes belong to the yield.*

In a constituent tree, D is called **immediate domination** and $\leq$ is called **domination**.

Definition 5 *An* **(unlabelled) dependency tree** *is a tree in which all nodes belong to the yield.*

In a dependency tree, D is called **government**, and $<$ is called **command.**[3] **Dependence** is the inverse of command.

Constituent trees and dependency trees are not exhaustive classes: trees do exist in which some but not all nonterminal nodes are yield nodes. Constituent trees and dependency trees are also not exclusive classes: a tree with only one node has no nonterminal nodes, hence is vacuously both a constituent tree and a dependency tree.

2.2 Labelled trees

Labelled trees are trees with labelled nodes and arcs.

Definition 6 *A* **labelling** *of the nodes of a graph* $G = (N, \sigma)$ *over a label-set* Z *is a partial function from* N *to* Z*. A labelling of the arcs of* G *is a partial function from* D_G *to* Z*.*

Node labels are used to assign syntactic categories and words to nodes, and arc labels are used to represent syntactic *roles* such as *head of*.

Definition 7 *A* **labelled tree** *over a label-set* (V, W, R) *is a structure* $t = (s, v, \lambda, \rho)$ *in which* s *is a tree,* v *is a function that labels each node of* s *with a category in* V*,* w *is a function that labels all and only the yield nodes of* s *with a word in* W*, and* ρ *is a partial function that labels some arcs of* s *with roles in* R*.*

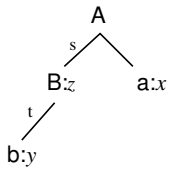
[3] Tesnière is not so careful to distinguish government and command as I am doing here, but I think the distinction is useful and appropriate.

Note that a labelled tree over (V, W, R) is also a labelled tree over (V', W', R) for all $V' \supseteq V, W' \supseteq W, R' \supseteq R$.

A **labelled constituent tree** is a labelled tree constructed on an unlabelled constituent tree and a **labelled dependency tree** is a labelled tree constructed on an unlabelled dependency tree. Henceforth, we use *constituent tree* and *dependency tree* to refer to labelled, not unlabelled, constituent and dependency trees.

In graphical representations of labelled trees, we separate category and word labels by a colon and place role names next to the arcs they label. Here is an example:

(2)



A, B, a, and *b* are categories, *x, y,* and *z* are words, and *s* and *t* are roles. The lexicon function λ is undefined for (nonyield) node *A,* and the role function ρ is undefined for the arc from *A* to *a.* The yield consists of the nodes *b, B, a,* in that order; the yield string is *yzx.* Note also that, since self-successors are distinguished by the presence of the word label, successor sets are unambiguously indicated in graphical representations such as (2), under the convention that precedence relations between self-successors and their children are indicated by the slant of the arcs connecting them to their children. For example, the successors of B in (2) are (b, B).

The structural properties of a tree are summed up in its root category, its grammatical productions, and its lexical productions, which we now define. The root category is simply the category of the root node. Each node of a tree has a grammatical production and a lexical production. If x has expansion $x \rightarrow y_1 \dots y_m[\cdot z_1 \dots z_n]$, its **grammatical production** is $v(x) \rightarrow^{\rho(x,y_1)} v(y_1) \dots ^{\rho(x,y_m)} v(y_m)[\cdot^{\rho(x,z_1)} v(z_1) \dots ^{\rho(x,z_n)} v(z_n)]$. For example, the root node of (2) has grammatical production $A \rightarrow^s Ba$ and the B node has grammatical production $B \rightarrow^t b\cdot$. Note that a terminal node x has grammatical production $v(x) \rightarrow \cdot$, and that no node has a grammatical production with an empty right-hand side (since trees are proper graphs). The **lexical production** of a node x is $v(x) \rightarrow w(x)$, if $w(x)$ is defined—which is to say, if x is a yield node—and $v(x) \rightarrow \emptyset$, otherwise. For example, the B node of (2) has lexical production $B \rightarrow z$. We write $S(t)$ for the singleton set containing the root category of t, $P(t)$ for the set of grammatical productions of nodes in t, and $\Lambda(t)$ for the set of lexical productions of nodes in t.

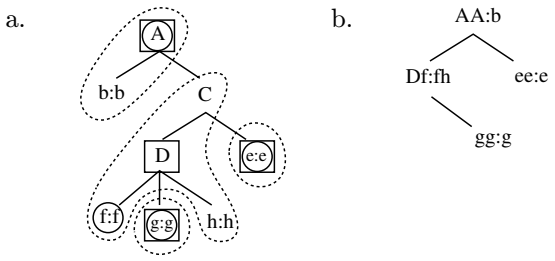
Definition 8 *A* **headed constituent tree** *is a constituent tree with a single arc-type H such that there is a unique outgoing arc labelled H from each nonterminal node.*

For a node x in a headed constituent tree, the child whose arc is labelled H is the **head child** of x. The **left children** of x are those children that precede the head, and the **right children** of x are those children that follow the head. If $(x_1, \ldots, x_n)$ is a sequence in which x_{i+1} is the head child of x_i, and x_n is a terminal node, then x_n is the **terminal head** of x_i for all $1 \leq i \leq n$.

2.3 Partition trees

A concept that will play a pivotal role in the rest of the paper is that of partition trees. Before launching in to definitions, let us consider a concrete example:

(3)



The sets of nodes surrounded by dotted lines in (3a) represent *partitions* of the tree; partitions are disjoint and every node belongs to some partition. The nodes in boxes are the *roots* of partitions, and the nodes in circles are the *heads* of partitions.

(3b) shows the *partition tree* of (3a). Each node of (3b) represents a partition in (3a). Ancestry relations in (3b) are inherited from roots of partitions, and precedence relations are inherited from heads of partitions. The category of a node in (3b) is the concatenation of the root and head categories of the corresponding partition, and the word of a node in (3b) is the concatenation of words in the corresponding partition.

For example, the root of the middle partition is D and its head is f. The A partition is parent of the D partition because the A node is the nearest partition-root that is ancestor of D, the root of the D partition. For the same reason, the A partition (and not the D partition) is the parent of the e partition. Because f precedes g, the partition D precedes its child-partition g; this is indicated by the slant of the line connecting the *Df* node in (3b) to the *gg* node.

In (3b), *Df*, *gg*, and *ee* are precedence-ordered, in that order. *Df*, *gg*, and *ee* constitute the yield; AA is the only node not in the yield. (3b) is a "mixed" tree in which some nonterminals (namely, *Df*) belong to the yield, but others (namely, AA) do not.

The example (3) gives some hints about what partitions are good for. For instance, if we think of the *Df* node in (3b) as containing the "amalgamation" of terminal nodes f and h from the original tree, appearing in the position of f, the effect is much the same as head-raising in GB. Again, readers familiar

with May's distinction between "categories" and "segments" may recognize an analogue in partition trees. If we take e to be adjoined to D in (3a), then (3b) represents the "domination" relation among categories. But it is premature to explore these issues here; we will return to them later.

Let us now define partition trees formally.

Definition 9 *A* **partitioning** *of a tree $t = (N, \sigma)$ is a structure $(\mathcal{P}, r, h)$ in which $\mathcal{P} = \{p_i\}$ is a set of* **partitions** *that are individually connected subgraphs of t, pairwise disjoint, and whose union is N; and r and h are functions assigning a* **(nominal) root** *and a* **head***, respectively, to each partition.*

We distinguish between the *nominal root* assigned by r and the **absolute root** of p, which is that node in p that is ancestor of every node in p. (We know it exists because otherwise p would not be connected.) Henceforth, the *root* of a partition is to be understood as the nominal root. when it is needed.

Under certain conditions, a partitioning $\pi = (\mathcal{P}, r, h)$ of a tree t defines a new tree whose nodes are partitions in $\mathcal{P}$. Let p and q be partitions in P. We define q to be a *proper p-descendant* of q in t just in case $r(q)$ is a proper descendant of $r(q)$. We define q to be a *p-child* of p iff it is an immediate proper p-descendant of p. Partition p is a *p-self-successor* just in case $r(p) < h(p)$—that is, just in case its root is a proper ancestor of its head or its root is a self-successor and identical to its head. The *p-successors* of p are its children, plus itself if it is a p-self-successor. We define p to *p-precede* q just in case $h(p) \prec h(q)$.

If, for every partition p, $r(p) \leq h(p)$ and the p-successors of p are totally ordered by p-precedence, then the partitioning is said to be **valid**. If a partitioning is valid, we can define the *p-successor set* of a partition p to be the sequence consisting of the p-successors of p ordered by p-precedence.

Definition 10 *Given a tree t and a valid partitioning $\pi = (\mathcal{P}, r, h)$ of t, the* **partition graph** *$d_\pi(t)$ of t is the graph $(\mathcal{P}, \sigma)$ where, for $p \in \mathcal{P}$, $\sigma(p)$ is the p-successor set of p in t.*

If a partitioning π is valid, has exactly one p-root, and all terminal partitions in π are self-p-successors, then π is said to be a **proper partitioning**.

Theorem 2 *The partition graph $d_\pi(t)$ is a tree iff π is a proper partitioning.*

Proof. Necessity is immediate from the definition of labelled tree. To show sufficiency, we must show that $d_\pi(t)$ is (1) proper, (2) single-rooted, (3) single-parented, and (4) acyclic. (1) holds by the definition of p-successor set (no duplicates) and the definition of proper partitioning (terminal nodes are self-successors). (2) follows from the other part of the definition of proper partitioning. (3) is true because each partition root has at most one greatest ancestor that is a partition root. (4) is inherited from the acyclity of the original tree, since $p \leq_d q$, for $d = d_\pi(t)$, only if $r(p) \leq_t r(q)$. ∎

To extend the notion of *partition tree* to labelled trees, we need to define the labels in a partition tree. Given a labelled tree $t = (s, v, w, \rho)$ and a proper partitioning π of s, we define the category of partition p to be $v_\pi(p) = v_t(r(p))v_t(h(p))$, that is, the concatenation of the category of the root of p and the category of the head of p. The word label of p is defined iff p is a p-self-successor; if p is a p-self-successor its word label is $w_\pi(p) = w_t(x_1)\ldots w_t(x_n)$ for $x_1, \ldots, x_n$ the yield nodes belonging to p, in order of precedence. The role label of the arc (p, q) is determined as follows. Let y be the absolute root of q and let x be the parent of y. Then $\rho_\pi(p, q) = \rho_t(x, y)$. Intuitively, the role of the arc connecting two partitions in a partition tree is the same as the role of the arc connecting them in the original tree. The labelled partition tree $d_\pi(t)$ is the labelled tree $(d_\pi(s), v_\pi, w_\pi, \rho_\pi)$.

It is also useful to define the *expansions* and *productions* of a partitioning π of t. If p is a self-p-successor its expansion is $p \to \alpha \cdot \beta$ for α the p-children of p that p-precede p and β the p-children of p that p-follow p in t. If p is not a self-p-successor its expansion is $p \to \gamma$ for γ the p-successor set of p. Given a partition p with expansion $p \to q_1 \ldots q_m[\cdot r_1 \ldots r_n]$, the grammatical production of p is $v_\pi(p) \to^{\rho_p i(p,q_1)} v_\pi(q_1) \ldots^{\rho_p i(p,q_m)} v_\pi(q_m)[.^{\rho_p i(p,r_1)} v_\pi(r_1) \ldots^{\rho_p i(p,q_n)} v_\pi(q_n)]$. The lexical production of p is $v_\pi(p) \to w_\pi(p)$, if $w_\pi(p)$ is defined, and $v_\pi(x) \to \emptyset$, otherwise. $P(\pi)$ and $\Lambda(\pi)$ are the grammatical and lexical productions, respectively, of the partitioning π of t. $S(\pi)$ is the singleton set containing the p-category of the root partition of π. For any partitioning π of a tree t, and $d = d_\pi(t)$ the partition tree induced by π, it should be obvious that $S(\pi) = S(d)$, $P(\pi) = P(d)$ and $\Lambda(\pi) = \Lambda(d)$, by construction.

3 Grammars

We define the language $L(t)$ of a tree t to be $\{u : S(u) = S(t), P(u) \subseteq P(t), \Lambda(u) \subseteq \Lambda(t)\}$. The language of a tree intuitively represents every valid way of recombining the structural properties of t. We extend S, P, Λ, and L to sets of trees in the obvious way: for a set of trees T, $S(T) = \bigcup_{t \in T} S(t)$, $P(T) = \bigcup_{t \in T} P(t)$, $\Lambda(T) = \bigcup_{t \in T} \Lambda(t)$, and $L(t) = \bigcup_{t \in T} L(t)$.

A grammar is a collection of structural properties. More precisely, a grammar over label-sets (V, W, R) is a structure $G = (S, P, \Lambda)$ where $S \subseteq V$ is a set of root categories; P is a set of grammatical productions $X \to^{r_1} Y_1 \ldots^{r_m} Y_m[.^{s_1} Z_1 \ldots^{s_n} Z_n]$ for $X, Y_i, Z_j \in V$ and $r_i, s_j \in R$; and Λ is a set of lexical productions $X \to w$ for $X \in V$ and $w \in W$.

The **language** $L(G)$ of a grammar G is the set of trees $\{t : S(t) \subseteq S_G, P(t) \subseteq P_G, \Lambda(t) \subseteq \Lambda_G\}$. A tree t is **admitted** by G just in case $t \in L(G)$. A tree is admitted by G just in case its structural properties are included in G, that is, $t \in L(G)$ iff $L(t) \subseteq L(G)$. Hence also $\bigcup_{t \in L(G)} L(t) = L(G)$, which is to say, $L(L(G)) = L(G)$. Viewing L as a generating relation, $L(G)$ is a fixed point.

Theorem 3 *A language T is $L(G)$ for some grammar G iff $L(T) = T$.*

Proof. We have just noted that $L(T) = T$ if $T = L(G)$ for some grammar G. Conversely, if $L(T) = T$, then clearly the grammar $G = (S(T), P(T), \Lambda(T))$ is such that $L(G) = T$. ∎

Note that $S(L(G)) \subseteq S(G)$, $P(L(G)) \subseteq P(G)$, and $\Lambda(L(G)) \subseteq \Lambda(G)$, but it is not necessarily the case that $S(L(G)) = S(G)$ or $P(L(G)) = P(G)$ or $\Lambda(L(G)) = \Lambda(G)$. G may have spurious root categories or productions—root categories or productions that appear in no tree in $L(G)$. In particular, note that any ϵ-production in P is useless: no node in a tree has an empty successor set, since trees are proper graphs, hence no ϵ-production belongs to $P(t)$ for any tree t. This is not a significant restriction, however. We can represent an "empty" node by assigning it the "empty" word label e; that is, e is the identity element for the string concatenation operation. (Note that the lexical production $X \to e$ is distinct from $X \to \emptyset$; the latter is the lexical production of a node that has no word label.)

A **proper grammar** is one without spurious productions or root categories. If G is proper, then $S(L(G)) = S(G)$, $P(L(G)) = P(G)$, and $\Lambda(L(G)) = \Lambda(G)$.

The **nonterminal categories** of G are those categories appearing on the lefthand side of nontrivial grammatical productions of G, that is, grammatical productions whose righthand side is neither empty nor "$\cdot$". The **terminal categories** of G are those appearing on the lefthand side of trivial grammatical productions of G. Note that nonterminal and terminal categories are not necessarily disjoint. The **yield categories** of G are those categories appearing on the lefthand side of nontrivial lexical productions of G, that is, lexical productions whose righthand side is not $\emptyset$. The **nonyield categories** of G are those categories appearing on the lefthand side of trivial lexical productions.

Since all and only nonterminal nodes have nontrivial grammatical productions, any nonterminal node in a tree accepted by G must have a nonterminal category and any terminal node must have a terminal category. Since all and only yield nodes have nontrivial lexical productions, any yield node in a tree accepted by G must have a yield category and any nonyield node must have a nonyield category.

Definition 11 *A* **constituent grammar** *(or* **context-free grammar***) over* (V, W, R) *is a grammar* $G = (S, P, \Lambda)$ *over* (V, W, R) *such that the terminal categories of* G *are disjoint from the nonterminal categories of* G*, and the yield categories of* G *are identical to its terminal categories.*

Theorem 4 *If* G *is a constituent grammar, all trees in* $L(G)$ *are constituent trees.*

Proof. The category restrictions guarantee that all and only terminal nodes (in trees accepted by G) are yield nodes. ∎

Definition 12 *A* **dependency grammar** *(DG) is a grammar* $G = (S, P, \Lambda)$ *over* (V, W, R) *for some* V, W*, such that the set of nonyield categories of* G *is disjoint from both terminal and nonterminal categories.*

Theorem 5 *If G is a dependency grammar, all trees in $L(G)$ are dependency trees.*

Proof. Since nonyield categories (if any) appear on the lefthand side of no grammatical productions at all, they are spurious, and every node in a tree accepted by G is a yield node. ■

A grammar G is an **unlabelled** grammar just in case it is a grammar over $(V, W, \emptyset)$ for some V and W. An unlabelled grammar accepts only unlabelled trees. If not indicated otherwise, we henceforth assume unlabelled gramamrs.

Definition 13 *A* **headed grammar** *(or* **headed context-free grammar** *(HCFG)) is a constituent grammar over some (V, W, R) such that, for some role $h \in R$, in every nontrivial grammatical production there is exactly one category on the righthand side that has a role, and its role is h.*

Theorem 6 *If G is a headed grammar, all trees in $L(G)$ are headed trees.*

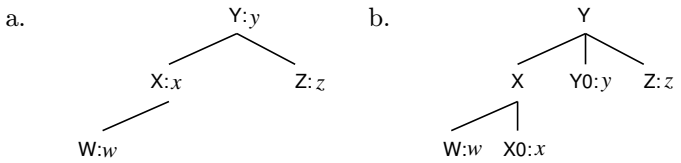
Proof. Obvious. ■

3.1 Gaifman's Equivalence

Gaifman defines a strong equivalence between dependency grammars and constituent grammars by defining the constituent tree *induced by* a dependency tree, and thence the constituent tree language induced by a dependency grammar. He proceeds to show that there are context-free tree languages that are not inducable by any dependency grammar.

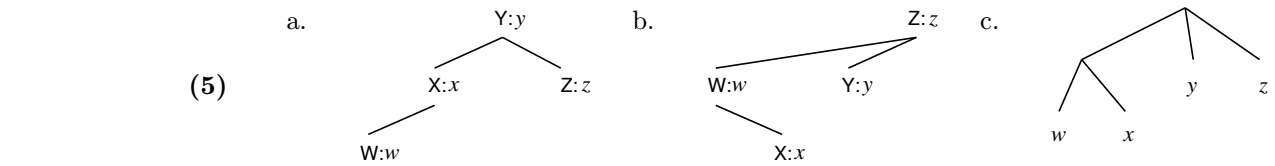
Under Gaifman's definition, to construct a constituent tree from a dependency tree, one introduces an extra terminal node for each nonterminal node in the dependency tree. The extra terminal node is placed between the left and right children, and bears the word label of the original nonterminal node. It is given a new category label. For example, the dependency tree (4a) induces the constituent tree (4b). $X0$ and $Y0$ are the new terminal nodes corresponding to the original nonterminal nodes X and Y.

(4)



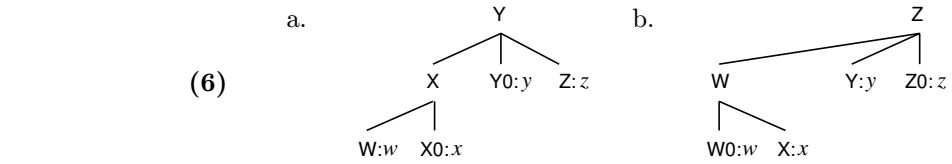
This may seem innocuous enough, but it has serious consequences. Gaifman proceeds to define two grammars to be strongly equivalent if they generate/induce the same set of *unlabelled* trees. But as Gaifman himself points out, dependency trees that differ not only in labels, but also in structure, induce the same unlabelled constituent trees. For example, the dependency trees (5a) and (5b) induce the same unlabelled constituent tree (5c).

(5)



Gaifman justifies his decision by pointing out that different dependency trees induce different *labelled* constituent trees. For example, the dependency trees in (5) induce the labelled trees in (6).

(6)



But there are two problems with this justification. First, Gaifman bases his results about the limited strong generative power of dependency grammars on unlabelled trees. And second, though the labelled trees in (6) are indeed distinct, it is a distinction without a difference: they are isomorphic. We have informally encoded head information in (6) in the vertical lines and the convention that $\alpha 0$ is the category of the head of α. But that information is nowhere present in the structure itself. In the actual constituent tree represented by (6a), there is nothing to distinguish the *X–X0* arc from the *X–W* arc, and X, $X0$, and W are simply three different categories; they could be renamed A, B, and C, and nothing essential would be changed.

As a result, Gaifman's result may be nothing more than an artifact of the way he defines the constituent trees induced by a DG. To make a meaningful comparison between DG's and CFG's with regard to their strong generative capacity, we require common ground that does not suppress essential features of either. Fortunately, we can find such common ground in headed constituent grammars. First we define headed constituent grammars, and then we show how they mediate between unheaded constituent grammars and dependency grammars.

3.2 Characteristic Grammars

There is a transparent relationship between headed and unheaded constituent trees: there is a unique unheaded constituent tree underlying the headed constituent tree; and similarly for headed and unheaded constituent grammars.

Definition 14 *The* **c-characteristic tree** *of a headed constituent tree* $t = (s, v, w, \rho)$ *is the unheaded constituent tree* $C(t) = (s, v, w, \emptyset)$.

The "c–" is meant to suggest "constituent".

We extend C to tree sets in the obvious way: $C(T) = \{C(t) | t \in T\}$.

Definition 15 *The* **c-characteristic grammar** *of a headed constituent grammar G is the unheaded constituent grammar $C(G)$ obtained by replacing each nontrivial grammatical production $X \to \alpha^h Y \beta$ of G with the corresponding unheaded production $X \to \alpha Y \beta$.*

Theorem 7 *For every headed constituent grammar G, $C(L(G)) = L(C(G))$.*

Proof. $C(G)$ is the constituent grammar that results from deleting head markings in productions. Since head markings affect only arc labels, the trees of $L(G)$ and $L(C(G))$ are identical except for arc labels, which are absent in $L(C(G))$. Deleting the arc labels in $L(G)$ yields $C(L(G))$, which is therefore identical to $L(C(G))$. ∎

Theorem 8 *Every HCFG has a unique c-characteristic grammar, and every CFG is the c-characteristic grammar of at least one HCFG, but not generally of a unique HCFG.*

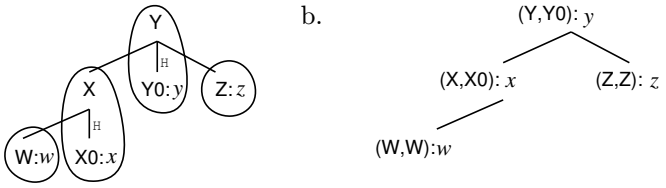
Proof. It is obvious that translating headed to unheaded productions yields a unique and well-formed CFG for every HCFG. In the reverse direction, consider an arbitrary CFG G. If we replace each nontrivial production of G with a headed production in which the first child is head, we have an HCFG H such that $C(H) = G$. This shows that every CFG is the c-characteristic grammar of some HCFG. However, there is generally more than one way to choose heads, hence more than one HCFG with the same characteristic grammar. For example, the HCFG H_1 with sole production $S \to^h a\ b$ and the HCFG H_2 with sole production $S \to a\ ^h b\cdot$ are distinct, yet $C(H_1) = C(H_2)$. ∎

Let us say that two HCFG's are **c-equivalent** iff they have isomorphic c-characteristic grammars. (Two grammars are isomorphic iff they are identical up to renaming of labels; I trust no confusion will arise if I omit the details.) By theorem 7, two c-equivalent HCFG's are strongly equivalent in the sense that they generate the same set of unheaded constituent trees, that is, $C(L(H_1)) = C(L(H_2))$ if H_1 and H_2 are c-equivalent. (If H_1 and H_2 are proper grammars, we can strengthen this to "iff".) In short, each CFG represents a class of strongly-equivalent HCFG's, and every HCFG belongs to exactly one such equivalence class.

3.3 Projection Trees

The relationship between headed constituent grammars and dependency grammars is not quite as obvious as that between headed and unheaded constituent grammars, but I think not less natural. The key is to map *projections* in headed trees to nodes in dependency trees, where a projection is a maximal sequence of nodes in which each is the head child of the next. For example, consider the headed constituent tree (7a), whose projections are circled. The relationships among the projections can be represented by a dependency tree (7b) whose nodes represent entire projections of (7a):

(7)



Definition 16 *An* r-**path** *in a labelled tree* t *is a sequence* $(x_1, \ldots, x_n)$ *such that* (x_{i-1}, x_i) *is an arc with role* r*, for all* $1 < i \leq n$*.*

An r-path is a connected, linear subgraph of t containing only arcs labelled r. Paths are also monotonic in the sense that they do not go up the tree and down again—no path contains more than one node out of a set of siblings.

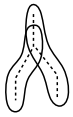
Definition 17 *An* r-**projection** *is a maximal* r*-path.*

Note that I do *not* use the term *projection* to refer to nodes in a path (as it is sometimes used); it always refers to the path itself. Also, projections are by definition maximal. The role "r–" will be omitted if it is clear from context—in particular, when headed constituent trees are involved, the only projections possible are h-projections, for h the head role.

The **root** of a projection p is the node in p with no arc leading to it from another node in p, and the **head** of p is the node in p with no arc leading from it to another node in p. Since paths are linear and monotonic, root and head exist and are unique for every path, and, in particular, for every projection.

Theorem 9 *(1) If every node in a labelled tree* t *has at most one* r*-child, then the* r*-projections of* t *partition* t*. (2) If every node has at least one* r*-child, then the head of each* r*-projection is a terminal node.*

Proof. (1) Since paths are monotonic (and trees have no re-entrancies), the only way that two paths can overlap is in the form:



If no node has multiple r-children, this situation cannot arise, ergo no projections overlap. Since every node belongs to a projection (in a pinch, to the trivial projection consisting solely of the node in question), projections partition the tree.

(2) If every nonterminal node has at least one r-child, then any r-path ending in a nonterminal node can be extended downward; hence r-projections—maximal r-paths—do not end until they reach a terminal node. ∎

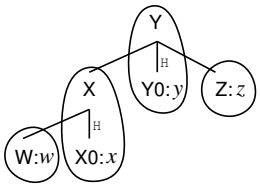
Corollary 1 *Let $\mathcal{P}$ be the set of projections of a headed constituent tree t, and let $r(p)$ and $h(p)$ be the root and head, respectively, of projection $p \in \mathcal{P}$. Then $\phi(t) = (\mathcal{P}, r, h)$ is a proper partitioning of t, and the partition tree $D(t) = d_{\phi(t)}(t)$ is a dependency tree.*

Proof. In a headed constituent tree t, every nonterminal node has a head-child, and the head-child is unique. Therefore, by theorem 9, the projections of t partition the nodes of t, and the head of every projection is a terminal node. The partitioning $\phi(t)$ is obviously valid; and since the root node of t is the root of a partition and for every terminal partition p, $r(p) < h(p)$, $\phi(t)$ is also proper. Hence $D(t) = d_{\phi(t)}(t)$ exists. For any partition p whose root is nonterminal, $r(P) < h(P)$, so all nonterminal nodes in $D(t)$ are self-successors, thus $D(t)$ is a dependency tree. ∎

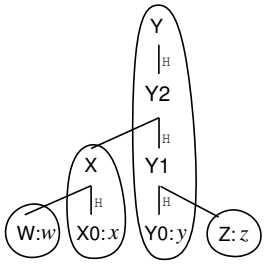
We call the dependency tree $D(t)$ induced by the projections of a headed tree t the **projection tree** of t. The projection tree of a headed tree is unique. But more than one headed tree may have the same projection tree. For example, (7b) is the projection tree of both (8a = 7a) and (8b).

(8)

a.



b.



Informally speaking, projection trees abstract away from the manner in which dependents are combined with their governor, including information about the order in which dependents are attached, and information about intervening unary-branching nodes that add no dependents.

Just as the c-characteristic tree of a headed tree t is characteristic in the sense of representing just the constituency properties of t, the projection tree of t is characteristic in the sense of representing just the dependency properties of t. For that reason, we will also call projection trees **d-characteristic trees**.

We extend ϕ to languages T as $\phi(T) = \{\phi(t) : t \in T\}$. We can also extend D to languages. Here, however, a technical issue arises. Trees $D(t)$ are special in a way that has no bearing on their structure: their nodes are partitions from other trees. This is appropriate for $\phi(T)$, since it represents the sets of projections of trees in T, but inappropriate for D, which is meant to represent only the interprojection *structure* of trees of T. For this reason we define $D(T)$ to be the closure of $\{D(t) : t \in T\}$ under isomorphism. (Note that $C(T)$ is already closed under isomorphism.)

We observed earlier that, for any partitioning π of a tree t, and $d = d_\pi(t)$ the partition tree induced by π, we have $S(\pi) = S(d)$, $P(\pi) = P(d)$ and

$\Lambda(\pi) = \Lambda(d)$. In particular, for all headed trees t, $S(\phi(t)) = S(D(t))$, $P(\phi(t)) = P(D(t))$, and $\Lambda(\phi(t)) = \Lambda(D(t))$. Hence, if $T = L(T)$, it follows that $S(\phi(T)) = S(D(T))$, $P(\phi(T)) = P(D(T))$, and $\Lambda(\phi(T)) = \Lambda(D(T))$.

3.4 D-Characteristic Grammars

We can extend D to grammars as follows. Define the *paths* of an HCFG G to be all sequences of rules $(X_1 \to \alpha_1^h Y_1 \beta_1, \ldots, X_m \to \alpha_m^h Y_m \beta_m)$ such that $Y_i = X_{i+1}$ for all $1 \leq i < m$. The *projections* of G are the maximal paths of G. We write $\Phi(G)$ for the set of projections of a headed grammar G.

The *category* of grammar projection $q = (X_1 \to \alpha_1^h Y_1 \beta_1, \ldots, X_m \to \alpha_m^h Y_m \beta_m)$ is the concatenation $X_1 Y_m$. Write $(Z_1, \ldots, Z_k) = \alpha_1 \ldots \alpha_m$ for the sequence of left-child categories of rules of p, and $(Z_{k+1}, \ldots, Z_n = \beta_m \ldots \beta_1$ for the sequence of right children of p. Then $X_1 Y_m \to Z_1 A_1 \ldots Z_k A_k \cdot Z_{k+1} A_{k+1} \ldots Z_n A_n$ is a *production* derived from q iff all $Z_i A_i$ are projection categories of G. Since there may be multiple projection categories whose first component is Z_i, there may be multiple productions derived from the grammar projection q. Let us write $P(q)$ for the set of productions derived from q. If Φ is a set of grammar projections, $P(\Phi) = \bigcup_{q \in \Phi} P(q)$ is the set of productions defined by Φ.

If t is a headed tree accepted by G and $\phi(t)$ are the projections of t, obviously every production in $P(\phi(t))$ is also in $P(\Phi(G))$, by construction. The converse actually may not hold if G is improper. But if G is proper, then every grammar projection of G, that is, every member of $\Phi(G)$, is attested in some tree of $L(G)$. Now if grammar projection $q \in \Phi(G)$ is represented by projection p in some tree in $L(G)$, the production $P(p)$ derived from the tree projection p corresponds to only one of the productions $P(q)$ derived from the grammar projection q. Suppose $P(p) = (XY \to Z_1 A_1 \ldots Z_k A_k \cdot Z_{k+1} A_{k+1} \ldots Z_n A_n$. If $Z_i B$ is also a projection category of G, then substituting $Z_i B$ for $Z_i A_i$, or doing multiple such substitutions, yields an alternative production $P' \neq P(p)$ in $P(q)$. But if G is proper, every alternative category $Z_i B$ is represented by some projection p' in some tree in $L(G)$, and excising the subtree rooted at the root of p' and substituting it for the corresponding child of p yields a tree of $L(G)$. Hence every production in $P(q)$ is attested in some tree in $L(G)$, and since q was an arbitrary grammar projection in $\Phi(G)$, it follows that every production in $P(\Phi(G))$ is attested. In short, $P(\phi(L(G))) = P(\Phi(G))$, and since $P(\phi(T)) = P(D(T))$ whenever $T = L(T)$, it follows that $P(D(L(G))) = P(\Phi(G))$.

In a similar manner, we define the lexical productions $\Lambda(\Phi(G))$ to be those productions $XY \to w$ such that XY is a grammar-projection category of G and $Y \to w$ belongs to Λ_G. And we define $S(\Phi(G))$ to be the set of grammar-projection categories XY of G such that X belongs to S_G. Clearly, if $\phi(t)$ are the projections of some tree $t \in L(G)$, then $S(\phi(t)) \subseteq S(\Phi(G))$ and $\Lambda(\phi(t)) \subseteq \Lambda(\Phi(G))$; and if G is proper then $S(\phi(L(G))) = S(\Phi(G))$ and $\Lambda(\phi(L(G))) = \Lambda(\Phi(G))$. Therefore, $S(D(L(G))) = S(\Phi(G))$ and $\Lambda(D(L(G))) = \Lambda(\Phi(G))$.

Definition 18 *The* **d-characteristic grammar** *of a headed constituent grammar* $G = (S, P, \Lambda)$ *is* $D(G) = (S(\Phi(G)), P(\Phi(G)), \Lambda(\Phi(G)))$*, provided that* $P(\Phi(G))$ *is finite.*

If G is a grammar over (V, W, R) then $S(\Phi(G))$ is no larger than $|V \times V|$ and $\Lambda(\Phi(G))$ is no larger than $|V \times V \times W|$, hence both are finite. However, if there is a cycle among the grammar projections of G, there may fail to be an upper bound on the length of righthand sides of productions derived from grammar projections, hence no upper bound on their number. But if $P(\Phi(G))$ is not finite, $D(G)$ is not a grammar. The **degree** of a headed grammar G is the number of children of its longest grammar projection, or infinite, if the number of children is unbounded.

Theorem 10 *For every finite-degree HCFG* G*,* $L(D(G)) = D(L(G))$*.*

Proof. Since G is of finite degree, $D(G)$ is defined. Let G' be the grammar obtained from G by eliminating all spurious productions and root categories. Then $L(G') = L(G)$ and $L(D(G')) = L(D(G))$. Since G' is proper, we know that $S_{D(G')} = S(\Phi(G')) = S(D(L(G')))$, $P_{D(G')} = P(\Phi(G')) = P(D(L(G')))$, and $\Lambda_{D(G')} = \Lambda(\Phi(G')) = \Lambda(D(L(G')))$. Hence $L(D(G)) = L(D(G')) = D(L(G')) = D(L(G))$. ∎

Theorem 11 *Every finite-degree HCFG has a unique d-characteristic grammar, and every DG is isomorphic to the d-characteristic grammar of at least one HCFG, but not generally a unique HCFG.*

Proof. That every finite-degree HCFG has a unique d-characteristic grammar is obvious by construction. In the other direction, given an arbitrary DG G, we can construct an HCFG G' such that G is isomorphic to $D(G')$ as follows. For each category X of G, X is a category of G' and so is a new "head" category corresponding to X, which we write $X0$. For each grammatical production $X \to \alpha \cdot \beta$ of G, G' has the production $X \to \alpha^h X0\beta$. For each lexical rule $X \to w$ of G, G' has the rule $X0 \to w$. Since all grammar projections of G' are trivial, consisting of single rules, it is straightforward to see that $D(G')$ is identical to G except having categories $(X)(X0)$ in place of X—that is, $D(G')$ is isomorphic to G.

Finally, to see that different HCFG's may have the same d-characteristic grammar, consider the HCFG G_1 with production $S \to^h A$ and lexical rule $A \to a$, and HCFG G_2 with productions $S \to^h B, B \to^h A$ and lexical rule $A \to a$. $D(G_1)$ and $D(G_2)$ are the same, namely, the grammar with sole grammatical production $SA \to$ and lexical rule $SA \to a$. ∎

We say that two HCFG's are **d-equivalent** iff they have isomorphic d-characteristic grammars. By theorem 10, d-equivalent grammars have isomorphic languages.

3.5 Gaifman's Result

As we have seen, both C and D induce equivalence classes of HCFG's. But the equivalence classes defined by C and D are incomparable. There are HCFG's that have the same c-characteristic grammar, but different d-characteristic grammars. Conversely, there are HCFG's with the same d-characteristic grammar, but different c-characteristic grammars. (9a) provides an example of the former; (9b) of the latter.

(8) a.

G	$S \to *a\ b$	$S \to a\ *b$
$C(G)$	$S \to a\ b$	$S \to a\ b$
$\Pi(G)$	$aS(\epsilon; b)$	$bS(a; \epsilon)$

b.

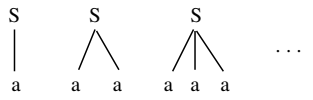
G	$S \to a\ *A$	$S \to *A\ c$
	$A \to *b\ c$	$A \to a\ *b$
$C(G)$	$S \to a\ A$	$S \to A\ c$
	$A \to b\ c$	$A \to a\ b$
$\Pi(G)$	$bAS(a; c)$	$bAS(a; c)$

In short, Gaifman was wrong to say that DG's are a special case of CFG's. It *is*, however, accurate to say that DG's are more limited than CFG's in a certain sense. Namely, the equivalence classes defined by CFG's include all the HCFG's, but the equivalence classes defined by DG's include only the finite-degree HCFG's. DG's do not permit a word to have unboundedly many dependents, but HCFG's do countenance that possibility.

Nonetheless, the difference is rather more technical than essential. In particular, suppose we replace the sequences of categories in the righthand side of productions and valencies with regular expressions. Context-free grammars thus generalized are known as **extended context-free grammars**. In an **extended HCFG**, rules are of the form $X \to \alpha \underline{Y} \beta$ where α and β are regular expressions over the set of categories. In an **extended dependency grammar** valencies have the form $X \to \alpha; \beta$ where α and β are regular expressions over the set of categories.

Some extended grammars generate tree-sets that are not generable by any simple grammar. For example, the set of trees

(9)



is generable by an extended constituent grammar with production $S \to a+$ but not by any simple constituent grammar. This is a fact about the language as a whole, however—any given tree in the language is a constituent tree, and any

given tree in the language of an extended constituent grammar is admitted by some simple constituent grammar.

The definitions of c-characteristic grammar and d-characteristic grammar can be generalized in the obvious way to extended grammars, and the results we have stated for simple grammars continue to hold for extended grammars. In particular, extended constituent grammars represent equivalence classes of extended HCFG's, and extended dependency grammars also represent equivalence classes of extended HCFG's, and these equivalence classes are incomparable. However, there is one important difference to the results with simple grammars.

Theorem 12 *For every extended HCFG G, $D(G)$ is an extended dependency grammar.*

Proof. The g-projections of G are of form $X_1 \to \alpha_1 \underline{Y_1} \beta_1, \ldots, X_n \to \alpha_n \underline{Y_n} \beta_n$, where α_i and β_i are regular expressions for all $1 \leq i \leq n$. It is obvious that the set of g-projections of G constitutes a regular language over productions of G (identify the lefthand side categories with states of a finite automaton). Hence, though the set of g-projections expanding g-category $\langle X, a \rangle$ is in general infinite, it can be described by a (finite) regular expression over productions, and the set of g-expansions of $\langle X, a \rangle$ is representable by a regular expression R over the α_i and β_i of the g-projections for $\langle X, a \rangle$, with categories in α_i, β_i replaced by g-categories. Since the α_i and β_i are regular expressions, composing them with R yields a regular expression R' over g-categories of G, which constitutes a righthand side for a valency of an extended dependency grammar. Moreover, there are only as many valencies as there are g-categories, which is to say, boundedly many. ∎