

CHAPTER SEVEN: (PRE)SEMILATTICES AND TREES

1 Informal Motivation

As we saw in Chapter 6 (Problem 8), given a CFG $\langle T, N, D, P \rangle$, a nonterminal $A \in N$, and a T -string $s \in C_A$, we can use the CFG to guide us in constructing a proof that $s \in C_A$. In fact, as anyone who has taken a course in formal language theory or computational linguistics will already realize, there are general procedures for deciding, given *any* CFG $\langle T, N, D, P \rangle$, *any* T -string s and *any* nonterminal A , whether or not $s \in C_A$. Such a procedure is called a **recognizer** because it tells, in effect, whether the CFG recognizes a given string as a member of a given syntactic category. In order to decide correctly that $s \in C_A$, the recognizer essentially must construct a *proof* that this is the case. What about making the correct decision when $s \notin C_A$? For that to be possible, the recognizer must in some sense ‘know’ when it has gotten to the point where, had there been a proof that $s \in C_A$, it would have found one; at that point it would render a negative decision. A **parser**, roughly speaking, is just a recognizer which renders not merely a decision but also (symbolic representations of) the proofs (if any) upon which the decision was based.

The construction of recognizers and parsers for CFGs and other kinds of formal grammars, one of the central concerns of both formal language theorists and of computational linguists, is a very highly evolved and subtle discipline, which unfortunately is beyond the scope of this book. However, the fundamental distinction between a parser and a (mere) recognizer has an analog that is relevant even for empirical/theoretical linguists (as opposed to formal language theorists and computational linguists), namely the intuition that a sentence is not just a string of words that belongs to C_S but rather a *way* that the string in question belongs to C_S . To take a very simple example, let’s consider a slightly expanded version of the toy English grammar in Chapter 6, as follows:

$$T = \{\text{Fido, Felix, Mary, barked, bit, gave, believed, heard, the, cat, dog, yesterday}\}$$
$$N = \{S, NP, VP, TV, DTV, SV, Det, N, Adv\}$$

D consist of the following lexical entries:

$$NP \rightarrow \{\text{Fido, Felix, Mary}\}$$

$VP \rightarrow \text{barked}$
 $TV \rightarrow \text{bit}$
 $DTV \rightarrow \text{gave}$
 $SV \rightarrow \{\text{believed, heard}\}$
 $\text{Det} \rightarrow \text{the}$
 $N \rightarrow \{\text{cat, dog}\}$
 $\text{Adv} \rightarrow \text{yesterday}$

P consists of the following PSRs:

$S \rightarrow NP VP$
 $VP \rightarrow \{TV NP, DTV NP NP, SV S, VP Adv\}$
 $NP \rightarrow \text{Det } N$

The only additions are (1) the nonterminal Adv (adverb); (2) the terminals **heard** and **yesterday**; (3) the lexical entries for **yesterday** as an adverb and for **heard** as a sentential-complement verb; and (4) the PSR $VP \rightarrow VP Adv$.

Now consider the string $s = \mathbf{Mary\ heard\ Fido\ bit\ Felix\ yesterday}$. According to our grammar, $s \in C_S$ (the syntactic category of sentences), but few (if any) syntacticians would say that s is an English sentence! Rather, they would say that the word string s corresponds to *two different sentences*, one roughly paraphrasable as *Mary heard yesterday that Fido bit Felix* and another roughly paraphrasable as *Mary heard that yesterday, Fido bit Felix*. Of course, these two sentences mean different things; but more relevant for our present purposes is that we can also characterize the difference between the two sentences purely in terms of two distinct ways of *proving* that $s \in C_S$.

To understand this point, remember from Chapter 6 that the set of syntactic categories is (informally) defined by simultaneous recursive definition as follows:

1. (Base Clause) If $A \rightarrow t \in D$, then $t \in C_A$.
2. (Recursion Clause) If $A \rightarrow A_0 \dots A_{n-1} \in P$ and for each $i < n$, $s_i \in C_{A_i}$, then $s_0 \dots s_{n-1} \in C_A$.

Then the two proofs run as follows:

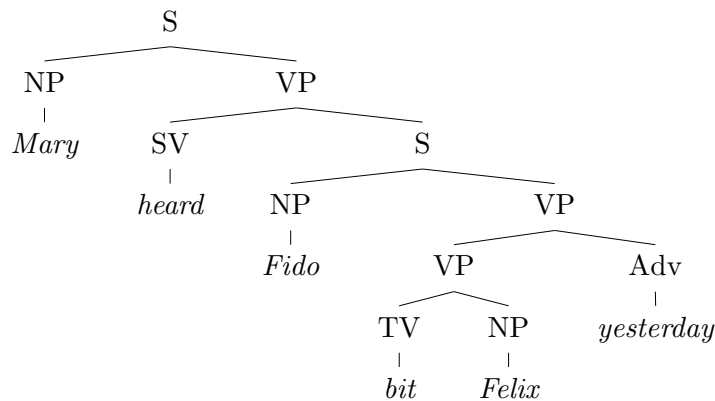
Proof 1: From the lexicon and the base clause, we know that **Mary**, **Fido**, **Felix** $\in C_{NP}$, **heard** $\in C_{SV}$, **bit** $\in C_{TV}$, and **yesterday** $\in C_{Adv}$. Then, by repeated applications of the recursion clause, it follows that:

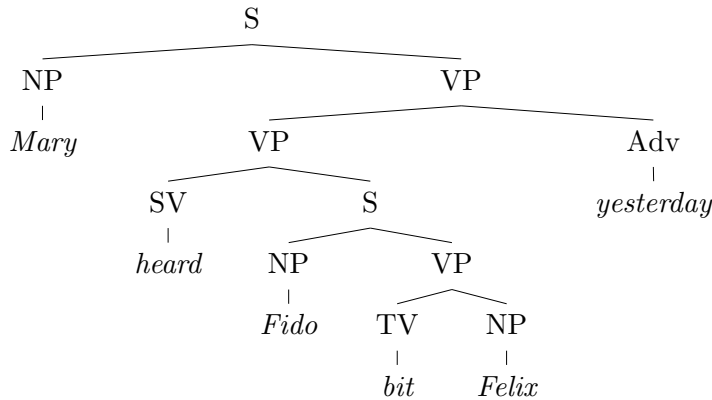
- a. since **bit** $\in C_{TV}$ and **Felix** $\in C_{NP}$, **bit Felix** $\in C_{VP}$;
- b. since **bit Felix** $\in C_{VP}$ and **yesterday** $\in C_{Adv}$, **bit Felix yesterday** $\in C_{VP}$;
- c. since **Fido** $\in C_{NP}$ and **bit Felix yesterday** $\in C_{VP}$, **Fido bit Felix yesterday** $\in C_S$;
- d. since **heard** $\in C_{SV}$ and **Fido bit Felix yesterday** $\in C_S$, **heard Fido bit Felix yesterday** $\in C_{VP}$; and finally,
- e. since **Mary** $\in C_{NP}$ and **heard Fido bit Felix yesterday** $\in C_{VP}$, **Mary heard Fido bit Felix yesterday** $\in C_S$.

Proof 2: The same as Proof 1, up through step a. From there, we proceed as follows:

- b. since **Fido** $\in C_{NP}$ and **bit Felix** $\in C_{VP}$, **Fido bit Felix** $\in C_S$;
- c. since **heard** $\in C_{SV}$ and **Fido bit Felix** $\in C_S$, **heard Fido bit Felix** $\in C_{VP}$;
- d. since **heard Fido bit Felix** $\in C_{VP}$ and **yesterday** $\in C_{Adv}$, **heard Fido bit Felix yesterday** $\in C_{VP}$; and finally,
- e. since **Mary** $\in C_{NP}$ and **heard Fido bit Felix yesterday** $\in C_{VP}$, **Mary heard Fido bit Felix yesterday** $\in C_S$.

There is nothing complicated about any of this, but this is not how a syntactician would usually describe the difference between the two homophonous sentences. Instead, s/he would draw two different *tree diagrams* as follows:





But what exactly *are* tree diagrams, and what is supposed to be their relationship to the linguistic phenomena being theorized about? Well, roughly speaking (we will get more precise in the following two sections), the diagrams are elaborated Hasse diagrams of mathematical objects called *labelled trees*. And what are labelled trees? Well, *trees* are partially ordered sets of a certain kind, and a labelled tree is a tree together with a function that assigns things called *labels* to the members (called *nodes*) of the partially ordered set. When syntacticians use labelled trees, the labels assigned to the minimal nodes are drawn from the set T of terminals and the labels assigned to the other nodes are drawn from the set N of nonterminals.

Intuitively, it is pretty clear that these two tree diagrams are closely related to, or in some sense correspond to, the two proofs given earlier (though the precise relationship remains quite obscure at this stage). But when we begin to construct a linguistic theory, which should we use? Should we use labelled trees or elaborations of them as set-theoretic idealizations of sentences, as is done in syntactic frameworks such as HPSG (head-driven phrase structure grammar) and LFG (lexical-functional grammar)? Or is it better to think of a sentence as a proof that a certain string belongs to a certain syntactic category, as is done in CG (categorial grammar)? Or is it perhaps best to use a hybrid approach with both set-theoretic and proof-theoretic aspects, such as most forms of transformational grammar (TG), which include MP (the Minimalist Program) and its predecessor GB (Government-Binding)? We will not be able to answer these questions until we start to formalize logic and look at how formal logic is applied to linguistic theory. But because tree representations are so widely used by syntacticians (not to mention semanticists, computational linguists, and logicians), it is important for us to get clear early on precisely what trees are and how syntacticians use them. To that end, it will be convenient to first consider

some somewhat more general notions, *(pre-)semilattices*, of which trees are a special case. These will turn out to have a wide range of other special cases, such as residuated lattices, heyting algebras, and boolean algebras, with important linguistic applications of their own.

2 LUBs, GLBs, and (Pre-)Semilattices

2.1 Least Upper Bounds and Greatest Lower Bounds

Throughout this section, \sqsubseteq is a preorder on a set A , and $\equiv (=_{\sqsubseteq})$ is the equivalence relation induced by the preorder.

Suppose $a \in A$ and $S \subseteq A$. Recall (Chapter 3, subsection 3.2) that a is a **greatest** (respectively, **least**) member of S provided, for every $b \in S$, $b \sqsubseteq a$ (respectively, $a \sqsubseteq b$). Greatest (least) members of A are called **tops** (**bottoms**). All greatest (least) members of S are equivalent. So if \sqsubseteq is an order, then S can have at most one greatest (least) member (because the equivalence relation induced by an order is the identity relation). In particular, in an order, there can be at most one top and at most one bottom.

Continuing to suppose $a \in A$ and $S \subseteq A$, recall also that a is a **maximal** (respectively, **minimal**) member of S iff, for every $b \in S$, $a \sqsubseteq b$ (respectively, $b \sqsubseteq a$) implies that $b \sqsubseteq a$ (respectively, $a \sqsubseteq b$), so that, in fact, $a \equiv b$. We observed that if \sqsubseteq is an order, then if a is the greatest (least) member of S , then it is also the unique maximal (minimal) member of S .

If $S \subseteq A$ and $a \in A$, we call a an **upper (lower) bound** of S provided, for every $b \in S$, $b \sqsubseteq a$ (respectively, $a \sqsubseteq b$). Thus by definition a greatest (least) member of S is an upper (lower) bound of S ; but the upper (lower) bounds of S need not be members of S . In case $S = A$, the notions of upper bound and greatest member (top) (lower bound and least member (bottom)) coincide.

The set of upper (lower) bounds of S is denoted by $\text{UB}(S)$ ($\text{LB}(S)$). In case S is a singleton $\{a\}$, $\text{UB}(S)$ ($\text{LB}(S)$) is written $\uparrow a$ ($\downarrow a$), read **up** of a (**down** of a).

A least member of $\text{UB}(S)$ is called a **least upper bound (lub)** of S , and a greatest member of $\text{LB}(S)$ is called a **greatest lower bound (glb)** of S . In case $S = A$, the notions of lub and top (glb and bottom) coincide. (A good exercise here is to find an example where a lub of S does not belong to S .) If S has a *unique* glb (lub), it is written $\sqcap S$ ($\sqcup S$). If A has a unique top (bottom), it is written \top (\perp). (A good exercise here is to show that

if there are any bottoms, they are the least upper bounds of \emptyset , and if there are any tops, they are the greatest lower bounds of \emptyset .)

If \sqsubseteq is an order on A and $S \subseteq A$, then S can have at most one glb (lub). (Why?) For the case $S = A$, this means there can be at most one top and at most one bottom. In the special case $S = \{a, b\}$, if S has a glb (lub), it is written $a \sqcap b$ ($a \sqcup b$). It is an easy but tedious exercise to show the following, for all $a, b, c \in A$:

Facts about \sqcap and \sqcup when \sqsubseteq is an order:

1. (Idempotence) $a \sqcap a$ exists and equals a ;
2. (Commutativity) if $a \sqcap b$ exists, so does $b \sqcap a$, and they are equal;
3. (Associativity) if $(a \sqcap b) \sqcap c$ and $a \sqcap (b \sqcap c)$ exist, they are equal; and
4. (Interdefinability) $a \sqsubseteq b$ iff $a \sqcap b$ exists and equals a .

Given these facts, one can immediately establish corresponding facts with all instances of \sqcap replaced by \sqcup , by considering A with the **opposite** (or **dual**) order \sqsubseteq^{-1} , usually called \sqsubseteq^{op} . This is an example of a widely used proof technique called **duality** whereby a result about a preorder is reinterpreted as a result about the dual (pre-)order.

Now suppose we have two sets A and B preordered by \sqsubseteq and \leq respectively. A function $f: A \rightarrow B$ is called **monotonic** or **order-preserving** with respect to the given preorders provided, for all $a, a' \in A$, if $a \sqsubseteq a'$, then $f(a) \leq f(a')$; and f is called **antitonic** or **order-reversing** with respect to the given preorders provided for all $a, a' \in A$, if $a \sqsubseteq a'$ then $f(a') \leq f(a)$. A monotonic (respectively, antitonic) bijection is called a **preorder isomorphism** (respectively, **preorder anti-isomorphism**) provided its inverse is also monotonic (respectively, antitonic). Two preordered sets are said to be **preorder-isomorphic** provided there is a preorder isomorphism from one to the other. Intuitively speaking, preorder-isomorphic preorders are “copies of each other”, differing only in which members they contain.

It is possible to consider to different preorders on the same set. For example, besides the usual order \leq on the nonzero natural numbers, we could also consider the order \sqsubseteq that holds between a pair of nonzero natural numbers if the first is a factor of (i.e. divides evenly into) the second.

If \sqsubseteq and \leq are two preorders on the same set A , we can ask whether the identity function on A is monotonic from the first to the second. Interestingly, many (all?) languages have a special grammatical construction, called the **correlative comparative** construction, to describe situations of this

kind. A typical English example is a sentence such as *The more expensive an SUV is, the more cupholders it has*, which asserts that the identity function on the set of SUVs is monotonic from \sqsubseteq to \leq , where, for two SUVs a and b , $a \sqsubseteq b$ means b costs at least as much as a , and $a \leq b$ means that b has at least as many cupholders as a .

2.2 (Pre-)semilattices

Now let A be a set with a preorder \sqsubseteq and a binary operation \sqcap (\sqcup) such that, for all $a, b \in A$, $a \sqcap b$ ($a \sqcup b$) is a glb (lub) of $\{a, b\}$ (not necessarily the only one). Such an operation is called a **meet** (**join**) operation, and a preorder equipped with such an operation is called a **lower** (**upper**) **presemilattice**; one with both is called a **prelattice**. If the preorder is an order, the *pre*-prefix is dropped: thus, an order with a meet (join) is a **lower** (**upper**) **semilattice**, and an order with both is simply a **lattice**. Then we have, for all $a, b, c, d \in A$:

Facts about \sqcap in a lower presemilattice:

1. (Idempotence up to Equivalence) $a \sqcap a \equiv a$;
2. (Commutativity up to Equivalence) $a \sqcap b \equiv b \sqcap a$;
3. (Associativity up to Equivalence) $(a \sqcap b) \sqcap c \equiv a \sqcap (b \sqcap c)$;
4. (Interdefinability) $a \sqsubseteq b$ iff $a \sqcap b \equiv a$;
5. (Monotonicity on Both Sides) For each $a \in A$, the function that maps each $b \in A$ to $a \sqcap b$ ($b \sqcap a$) is monotonic.
6. (Substitutivity up to Equivalence) If $a \equiv c$ and $b \equiv d$ then $a \sqcap b \equiv c \sqcap d$.

Duality gives corresponding facts for join in an upper presemilattice. We also have the following:

Facts about \sqcap and \sqcup in a prelattice:

- a. (Absorption up to Equivalence) $(a \sqcup b) \sqcap b \equiv b \equiv (a \sqcap b) \sqcup b$;
- b. (Semidistributivity) $(a \sqcap b) \sqcup (a \sqcap c) \sqsubseteq a \sqcap (b \sqcup c)$.

A prelattice is called **distributive** if the inequality reverse to Semidistributivity holds:

$$a \sqcap (b \sqcup c) \sqsubseteq (a \sqcap b) \sqcup (a \sqcap c)$$

holds. Thus in a distributive prelattice, we have the following (Distributivity up to Equivalence):

$$a \sqcap (b \sqcup c) \equiv (a \sqcap b) \sqcup (a \sqcap c)$$

It can be shown (though it is a fair amount of work) that this equivalence holds in a prelattice just in case the dual one (formed by interchanging meets and joins) does.

Of course all the equivalences stated in this section become equalities if the preorder in question is an order.

3 Trees

3.1 Technical Preliminaries

Here we gather together some facts that will simplify the discussion of trees.

Theorem 1 Any nonempty finite order has a minimal (and so, by duality, a maximal) member.

Proof sketch: Let T be the set of natural numbers n such that every ordered set of cardinality $n + 1$ has a minimal member, and show that T is inductive. The main idea of the proof is to show that T is an inductive set.

Corollary Any nonempty finite chain has a least (and so, by duality, a greatest) member.

Proof: This follows from the fact (itself a simple consequence of connectivity) that in a chain, a member is least (greatest) iff it is minimal (maximal).

Theorem 2 For any natural number n , any chain of cardinality n is order-isomorphic to the usual order on n (i.e. the restriction to n of the usual \leq order on ω).

Proof sketch: By induction on n . The case $n = 0$ is trivial. By inductive hypothesis, assume the statement of the theorem holds for the case $n = k$ and let A of cardinality $k+1$ be a chain with order \sqsubseteq . By the Corollary, A has a greatest member a , so there is an order isomorphism f from k to $A \setminus \{a\}$. The rest of the proof consists of showing that $f \cup \{< k + 1, a >\}$ is an order isomorphism.

Theorem 3 Suppose \sqsubseteq is an order on a finite set A . Then $\sqsubseteq = \prec^*$. That is: a finite order is the reflexive transitive closure of its own covering relation.

Proof: That $\prec^* \subseteq \sqsubseteq$ follows easily from the definition of reflexive transitive closure and the transitivity of \sqsubseteq . To prove the reverse inclusion, suppose $a \sqsubseteq b$ and let X be the (nonempty, finite) set of all subsets of A which, when ordered by \sqsubseteq , are chains with b as greatest member and a as least member. (X is nonempty since one of its members is $\{a, b\}$.) Then X itself is ordered by \subseteq_X , and so by Theorem 1 has a maximal member C . Let $n + 1$ be $|C|$; by Theorem 2, there is an order-isomorphism $f: n + 1 \rightarrow C$. Clearly $n > 0$, $f(0) = a$, and $f(n) = b$. Also, for each $m < n$, $f(m) \prec f(m+1)$, because otherwise, there would be a c properly between $f(m)$ and $f(m+1)$, contradicting the maximality of C .

3.2 Trees

We now define a **tree** is to be a finite set A with an order \sqsubseteq and a top \top , such that the covering relation \prec is a function with domain $A \setminus \{\top\}$. In the linguistic community, the following terminology for trees is standard:

1. The members of A are called the **nodes** of the tree.
2. \top is called the **root**.
3. If $x \sqsubseteq y$, y is said to **dominate** x ; and if additionally $x \neq y$, then y is said to **properly dominate** x .
4. If $x \prec y$, then y is said to **immediately dominate** x . In that case $y = \prec(x)$ is called the **mother** of x , and x is said to be a **daughter** of y .
5. Distinct nodes with the same mother are called **sisters**.
6. A minimal node (i.e. one with no daughters) is called a **terminal** node.
7. A node which is the mother of a terminal node is called a **preterminal** node.

We state here some important facts about trees, sketching some of the proofs and leaving others as exercises.

Theorem 4 No node can dominate one of its sisters.

Proof: Exercise.

Theorem 5 For any node a , $\uparrow a$ is a chain.

Proof sketch: Use the RT to define a function $h: \omega \rightarrow A$, with $X = A$, $x = a$, and F the function which maps non-root nodes to their mothers and the root to itself. Now let $Y = \text{ran}(h)$; it is easy to see that Y is a chain, and that $Y \subseteq \uparrow a$. It remains to show that $\uparrow a \subseteq Y$. So assume $b \in \uparrow a$; we have to show $b \in Y$.

By definition of $\uparrow a$, $a \sqsubseteq b$, and so by Theorem 3 of section 1, $a \prec^* b$. From this and the definition (Chapter 4, section 4) of reflexive transitive closure, it follows that there is a natural number n such that $a \prec_n b$, where \prec_n is the n -fold composition of \prec with itself. In other words, there is an A -string $a_0 \dots a_n$ such that $a_0 = a$, $a_n = b$, and for each $k < n$, $a_k \prec a_{k+1}$. But then $b = h(n)$, so $b \in Y$.

Corollary Two distinct nodes have a meet iff they are comparable.

Proof: Exercise.

Theorem 6 Any two nodes have a lub (and so a tree is a join semilattice).

Proof: Exercise.

3.3 Ordered Trees

An **ordered tree** is a set A with *two* orders \sqsubseteq and \leq , such that the following three conditions are satisfied:

1. A is a tree with respect to \sqsubseteq .
2. Two distinct nodes are \leq -comparable iff they are not \sqsubseteq comparable.
3. (No-tangling condition) If a, b, c, d are nodes such that $a < b$, $c \prec a$, and $d \prec b$, then $c < d$.

In an ordered tree, if $a < b$, then a is said to **linearly precede** b .

Theorem 7 If a is a node in an ordered tree, then the set of daughters of a ordered by \leq is a chain.

Proof: Exercise.

Theorem 8 In an ordered tree, the set of terminal nodes ordered by \leq is a chain.

Proof: Exercise.

4 Trees in Syntax

In many syntactic frameworks, ordered trees (often elaborated with some additional structure) are employed in the modelling of linguistic expressions (words and phrases). In such contexts, the trees are variously called *phrase structures*, *phrase structure trees*, or *phrase markers*. Typically (though the details differ from framework to framework, as discussed below), the grammar (in the sense of the total theory of the particular natural language being analyzed) will include a CFG $\langle T, N, D, P \rangle$, and the phrase structure trees will be ordered trees equipped with a **labelling** function that assigns terminal symbols to terminal nodes and nonterminals to nonterminal nodes. We then speak of the set of phrase structure trees **generated by**, or **licensed by**, or **admitted by** the CFG, which is defined as follows:

The Set of Phrase Structure Trees Admitted by a CFG

A phrase structure tree is **generated** by the CFG $\langle T, N, D, P \rangle$ iff

1. for each preterminal node with label A and (terminal) daughter with label t , $A \rightarrow t \in D$; and
2. for each nonterminal nonpreterminal node with label A and linearly ordered (as per Theorem 7) daughters with labels A_0, \dots, A_{n-1} respectively, ($n > 0$), $A \rightarrow A_0 \dots A_{n-1} \in P$.

Additionally, for a phrase structure tree with linearly ordered (as per Theorem 8) set of terminal nodes a_0, \dots, a_{n-1} with labels t_0, \dots, t_{n-1} respectively, the string $t_0 \dots t_{n-1}$ is called the **terminal yield** of the phrase structure tree.