

Robert C. Berwick

IS DNA A LANGUAGE?

Both DNA and what people speak are commonly referred to as *languages*. The analogy holds, at least in the formal sense. Both DNA and human languages encode and transmit information. Both, like beads on a string, form concatenative symbol-systems. Murkier by far is how much further down the scientific road this analogy can carry us. That is the question this chapter tries to answer: Is there indeed a "language of the genes"? Can linguistic science repair what Collado-Vides (chapter 9) correctly pinpoints as the weak link in current molecular biology—namely, the relative poverty of *explanatory* molecular biology, as opposed to *descriptive* molecular biology? Modern molecular biology's reductionism comes at a steep price, leaving us chock full of complex visibles but largely bereft of corresponding simple invisibles. Why do the bacterial sigma 70 and 54 promoters look this way rather than some other way? To be sure, evolution and physical science ultimately fix these answers. Even so, explanation-seeking scientists rightly posit intermediate, theoretical selections to account for such things as a quark's spin, an electron's valence or, more to the point, a person's genes.

Our initial question about genetic language—regarding whether linguistics can shed light on molecular biology—must have two simple answers: Yes, molecular biology can benefit from linguistic science (as Collado-Vides notes) simply by providing the right general scientific scaffolding, including modern linguistic theory's modularity principles and parameterized abstraction. Specifically, as we describe later, the mechanics of both molecular biology and natural languages are grounded on the notion of *adjacency* as a fundamental principle; there is no "action at a distance" (figure 15.1). Just as the intron and exon machinery pastes together previously disconnected pieces of genetic code into an adjacent working whole, grammatical relations such as the agreement between sentence subject and verb (in the figure, between the plural *s* ending on *the guys* and the *nonexistence* of an *s* on *like*) are defined only under strict adjacency, and almost the whole point of Chomsky's transformational grammar is to paste together previously disconnected sentence elements.

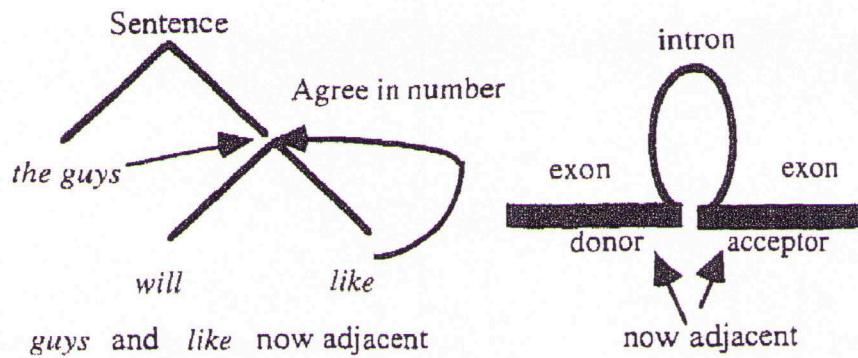


Figure 15.1 Both natural languages and the genetic language contain machinery that converts superficially "long-distance" relationships into adjacent ones.

However, no is an equally correct answer to our fundamental question because, as we shall see, natural languages form a much simpler computational system than the genetic code and transcription machinery. In a nutshell, whereas transcription exploits the three-dimensional twists and turns of biochemistry and resembles a general programming language (as noted in chapter 3), in contrast our current understanding is that natural language exploits *only* adjacency as its programming "trick." Adjacency is enough to derive (hence, explain) most of what we see in natural languages. No such corresponding explanation of why the genetic "programming language" looks the way it does has been forthcoming. The conclusion, then, is that the language of the genes is not like a natural language but more like a general programming language, the details of which we still do not fully understand. It is akin to looking at the input and output of a spreadsheet and, from that, trying to figure out not only the specific programming language instructions used but also which programming language was used—whether C, Fortran, or Pascal. As Lewontin notes in chapter 1, to understand this is probably the most difficult task of reverse engineering that anyone has ever undertaken. If this insight is accurate, it suggests that molecular biologists might do better to study the methods used by "clean-room" programmers to reverse-engineer spreadsheet programs than to try to figure out whether DNA or its transcription mechanisms generate certain kinds of non-context-free languages. More specifically, if we search through the space of context-free or non-context-free languages, we are simply searching through the *wrong* space. For natural languages, this is a space of restricted adjacency relations (described later). For the language of the genes, the appropriate representation is not yet clear, but it may be that something like the space of genetic "circuits" is more fitting (see chapters 6 and 13; McAdams and Shapiro, 1995).

The remainder of this chapter expands these points. First, we review the possible connection between the genetic code and formal language theory, showing that formal language theory serves as a poor proxy for studying programming languages and natural languages and hence is an unlikely candidate for investigating either one. The argument carries over to attempts to detect various patterns in the genetic code via different kinds of pattern-

matching languages. Here (as discussed in chapter 3) many popular algorithms are based on linear string matching, including so-called hidden Markov models. Though such linear models *have* been successful in mirroring some aspects of human language, it is crucial to observe that these linear models have largely been successful in modeling speech—that is, exactly that area of human language that is strictly linear and left-to-right. Second, we turn to the differences between genetic transcription and natural languages, demonstrating how much simpler natural languages are than DNA transcription. We also demonstrate that by using a more appropriate representation—defined over four natural configurations such as *subject*—one can build better search routines for natural language patterns. Finally, we argue that the language of the genes might best be expressible as a programming language or some such constraint system, perhaps like the genetic circuits discussed elsewhere in this book. This is an area for future research.

FORMAL, NATURAL, AND BIOLOGICAL LANGUAGES

Because DNA *is* a formal language, there is a natural temptation to wheel out the armamentarium of formal language theory, but can formal language theory help us understand DNA? To answer this question, one must first understand why formal language theory was invented. Elsewhere the argument has been made (Berwick, 1989) that it is rash to expect a complex biological system such as human language to abide by elegant mathematical rules such as those that define the Chomsky hierarchy of finite-state, context-free, context-sensitive, and Turing complete (arbitrary programming) languages. The Chomsky hierarchy itself is the wrong way to size up natural languages: Languages simply don't fall neatly into one of these classes. If that is so for human languages, then it is doubly so for DNA: Indeed, as we discuss later, although there is at least some new support for an elegant algebraic description for the "core" of natural language syntax, in this regard at least, DNA seems *more* complex than natural languages.

Formal Description of Transcription

There have been some efforts (see Searls, 1993, for a particularly illuminating and insightful study) to determine whether DNA, tRNA, their various substructures, or transcription machinery itself falls into one or another of the well-known formal language theory classes. To understand the results of such studies, we it would do well to recall both what formal language theory classes define and what role formal language theory played in aiding linguistic theory and in programming languages. Formal language theory was used in the 1960s to study both formal linguistics and the complexity of programming languages, but it has not been used much since then, because computer science has developed much keener methods for analyzing computational complexity.

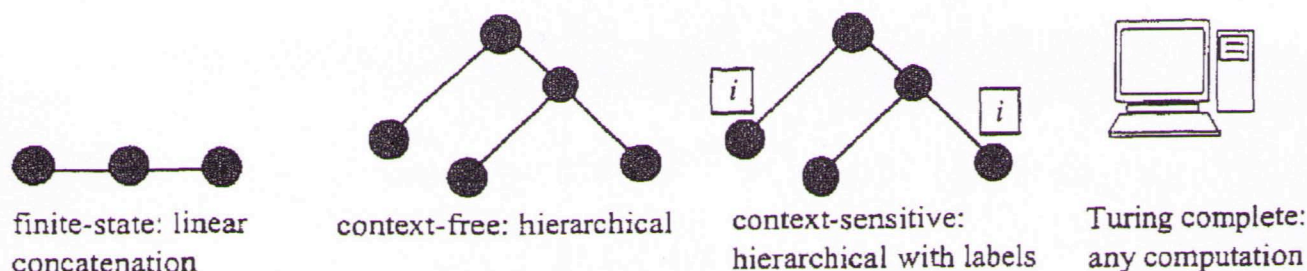


Figure 15.2 The Chomsky hierarchy: from linear to arbitrary (Turing complete) languages.

Formal Language Theory: A Brief History

Broadly speaking, formal language theory and especially the Chomsky hierarchy served as a rough *proxy* for particular complexity analyses and structural properties of both programming languages and linguistic relations. The hierarchy's relationship to computation itself is indirect. The hierarchy consists of four increasingly complex structural relations that define strict subsets of string classes (languages), as shown in figure 15.2: purely *linear concatenative* relations, or *finite-state languages*; purely nested or hierarchical, tree-like relations, or *context-free languages*; tree structures augmented with labeling tags that can refer to each other across arbitrary parts of the tree, or *context-sensitive languages*; and completely arbitrary relations or arbitrary programming languages, so-called *Turing complete languages*, that can compute anything that a general programming language (such as Fortran or C) can compute.

Used diagnostically, these classes are a blunt knife because human languages do not fall neatly into any one of these classes; for example, it is by no means clear that human languages need even be computable, in the strict sense, although presumably this is so. Natural languages certainly contain recursive, hierarchical structures or *phrases*, such as "the different types of RNA polymerases," in which the group of words of *RNA polymerases* is clearly a substructure that modifies *different types*—so natural languages are at least describable as context-free languages. Beyond this, however, this blunt taxonomy has yielded very few concrete results for linguistics. Chomsky (1956) more or less established that human languages cannot be contained in the class of finite-state languages. Similarly, Searls (1993, p. 73) shows that nucleic acids are more complex than simple linear finite-state languages: They encode palindromes, embeddings, and the like (with one technical caveat that we elucidate later). Though this is an interesting discovery about nucleic acids, and though it does suggest that pattern-matching techniques for analyzing sequences will have to do more than just look at linear models, again it is important to ask whether we gain by this any new insight. Searls (1993) himself notes that it does not gain us much. The real question is whether formal language theory could ever hope to tell us much.

The answer to this last question for linguistic theory has been plain. Beyond Chomsky's original discovery (1956), formal language theory has not

contributed substantially to our understanding of human language structure. Chomsky showed that linear analysis does not suffice to model human language; we need at least some notion of hierarchy. (In a later section, we show just what kind of hierarchy is required). The problem with going beyond this is that the formal language theory classes do not correspond to human languages. An infinite number of context-free (strictly hierarchical) languages are not natural languages, and these include sequences found in nucleic acids. For instance, consider the example that Searls (1993) uses to show that nucleic acid sequences are not purely linear, or finite-state: palindromes, or mirror-image, nucleic acid sequences. Such sequences are very easily generated by simple first-in, first-out push-down stacks—like placing a pile of dinner plates one on top of another and then removing the last one put on first—so one might expect to find such patterns, in the form $w_1 w_2 w_3 w_3 w_2 w_1$ or a *nested dependency*, in human languages. Instead, we find that a pattern more commonly found in natural languages, as in German or Dutch, is the *opposite* of push-down stack order—that is, the pattern $w_1 w_2 w_3 w_1 w_2 w_3$ or an *intersected dependency*. Evidently, this intersected pattern can also be found in some of the substructures of gene regulation. In this sense, nucleic acids patterns are *not* like natural languages—they contain more than do natural languages. Similarly, computer programming languages such as Fortran are syntactically context-free yet are certainly not natural languages; unlike natural languages, they require explicit instruction to learn, as any beginning programmer could tell you.

Of course, there also is no reason to believe that natural languages are some subset of the context-free languages. In hindsight, formal language theory turned out to be eminently helpful in describing programming languages but not natural languages. One might then wonder why formal language theory was wheeled out at all to attack the problem of natural languages. This appears to be simply an instance of the “lamplight fallacy”—looking where the mathematical light shines brightest—as is discussed elsewhere (Berwick and Weinberg, 1979): Researchers turned to formal language theory because it had clean mathematical properties and none of the unruly tangles of human language.

This aesthetic urge still surfaces even in the recent formal demonstrations about nucleic acids mentioned earlier. Most commonly, the argument runs this way: (1) We isolate some subset pattern in, say, English; (2) we show that this subset pattern has property *P*; and (3) we conclude therefore that English has property *P*. For example, for nucleic acid sequences, we might point out, as Searls does, that they contain palindrome sequences (step 1). Because palindromes cannot be generated by any finite-state or purely linear automaton, but can be generated as a strictly context-free language, we could, following step 2, identify *non-context-free* as the property *P* we want to isolate. Finally, according to step 3 of the argument, we conclude that nucleic acid sequences are not finite-state. However, this argument is flawed: Whereas this

subset of the nucleic acid sequences is not finite-state, it does *not* follow that the entire system is not finite-state. To explain further, note that the language of all possible nucleotide sequences of A, T, G, C—that is, the language Σ^* defined over the alphabet A, T, G, C—is certainly a regular or finite-state language but just as surely contains palindrome sequences, because it contains *all* possible sequences. To make the three-step argument apply, one must *intersect* the language studied—English or nucleic acid sequences—with some specially constructed filter designed to pick out just those sequences we know to be palindromes. Of course, this filter itself must be finite-state, and we must show that the filtering operation also is finite-state or regular (in the usual proofs, one can use set intersection as the filter because finite-state languages are closed under intersection), otherwise, we could introduce spurious non-finite-state complexity. To be sure, this point about subset properties is not easy to see. In fact, even Chomsky's original demonstration that English is not finite-state (1956) suffered from exactly this fallacy: Chomsky demonstrated that English contained patterns that were *not* finite state but did not precisely spell this proof out via intersection with an English-like "test pattern" so that the proof would apply formally. At least on first glance, then, Searls's demonstration (1993) that DNA is not context-free contains the same problems. However, it is usually easy to patch such proofs, so this is not meant as a damning critique. Rather, we should remain aware that it is too easy to single out mathematical purity at the expense of biological reality: Formal language theory does not naturally correspond to the theory of human languages, and we should not expect it to.

The Case of Hidden Markov Models

Another lesson to be learned from the lamplight fallacy relates to currently popular methods such as neural networks and hidden Markov models for discovering structure in sequence data. Here too one must be extremely careful in considering the assumptions about sequence or linguistic structure that these models make; otherwise, one will get back only what these models are able to find.

Consider the case of hidden Markov models (HMMs). These are a subcase of the finite-state languages (i.e., a linear sequence of states) but with the addition of *probabilities* on state transitions (which are hidden from our explicit view; hence the term) and associated probabilities on the actual output letters (e.g., the base alphabet) that are observed. The rough idea behind the HMM method is an update "learning" loop based on Bayes's rule: Start with some prior estimate of the hidden transition probabilities between states (say, a uniform one that assigns equal probabilities to all transitions) and then update those probabilities based on counting the sequences that are actually found, as opposed to those that are not found. (The exact method uses the Dempster-Shafer expectation maximization, or EM, algorithm.) After some

initial set of sequence data has been processed in this way, we arrive at some "final" estimate of the hidden state transitions, which then can presumably be used as a more accurate reflection of the "true" model underlying the sequence generation.

To understand what HMMs can buy us in both the linguistic and the molecular biology worlds, we must understand their limiting assumptions. First, HMMs make strict assumptions about the generative processes creating the observed nucleotide sequence—namely, that it is a linear, memoryless process. Clearly, this does not encompass the long-distance cutting and pasting of intron and exon machinery, let alone more complex transcription programs. Thus, HMMs can discover linear patterns or classifications, but we cannot expect them to discover the transcription "program" because HMMs cannot even represent such sophisticated properties. Further, the EM search method has its own limitations: It is a local-gradient-ascent, or hill-climbing system; the probability estimation algorithm tries locally to improve its current estimate based on where it currently is in a search space. Such an algorithm is guaranteed to find a maximum, or best estimate, but only a local maximum. If the search space contains sharp ridges or peaks, then the algorithm can get stuck there (one reason why heuristics and parallel search methods such as those described in chapter 3 often are appealed to).

Not surprisingly, then, for natural languages, HMMs have been most successfully used for precisely those representations that are linear—namely, sound sequences. They are used for speech recognition because, for the most part, one single articulated sound depends on just the one or two sounds preceding it. For more complex linguistic descriptions that go beyond local linear descriptions, HMMs perform much worse. For example, consider a sentence such as, "How many guys do you think were arrested?" (in which *guys* and *were* must both be plural, we say that they agree in number). Note the problem with a sentence that violates this constraint, such as "How many guys do you think was arrested?" Here, there is a relation between *guys* (the subject) and *were* or *was* (the verb) separated by a long distance. The whole point of modern transformational grammar (indeed, all modern grammatical theories) is to propose descriptive levels where these two elements are brought into adjacency (so that their features can be checked for agreement). In this case, Chomsky's modern transformational theory posits an unpronounced element (seen in the representation, but not heard) that serves as the object of *arrested* and is linked to *guys*: "How many guys_{*i*} do you think were arrested [empty]_{*i*}" (where the index *i* indicates the link). Using this representation the verb form *were* and the word *guys* are adjacent to one another. However, the operation that puts them together—a transformation—is not linear or local: The single transformational operation in current linguistic theory says that one can move a phrase such as *many guys* anywhere. This is beyond the descriptive power of HMMs, because HMMs, by definition,

describe *memoryless* processes and, in an example such as this, one has in effect “remembered” the position of the object of *arrested* so as to link it arbitrarily far away from the position where *many guys* actually is spelled out. Thus, we would not expect HMMs to provide a good discovery procedure for such linguistic relations.

One way to shore up the weaknesses of linear HMMs is to add some notion of hierarchical patterns. This has been affected in the basic EM algorithm and is used also in hierarchical pattern-matching algorithms; for natural languages, the analog is to use stochastic context-free grammars. However, here too one can show that these methods work mostly to the extent that the right hierarchical structure is prebuilt into them. The “topology” of the relations—what variable is linked to what other variable—must be understood in advance; otherwise, the search algorithm will not find the correct representation for us.

For instance, it can be shown that the EM algorithm simply will not find the right structure for a simple phrase such as *walking on air* (the true structure being a verb phrase in the form verb—prepositional phrase, with the prepositional phrase subtree consisting of the preposition *on* followed by the noun *air*). Instead, it will converge to a local minimum, wherein the verb is clustered erroneously with the preposition as a unit, apart from the noun *air*. If one examines more closely just why this is so, it turns out that the context-free rule space is not searched completely by the HMM algorithm; instead, there are two “peaks” or local maxima that force the system to cluster either the verb with the preposition first (the wrong result) or the preposition with the noun (the right result), and most of the space leads one to the first, erroneous conclusion. This search space is simply the wrong one to look at. Put another way, context-free rules seem to be the wrong representation to describe the linguistic relations in this case, and therefore no amount of clever searching can repair the representational defect. The right move is to use the correct representation from the start, to say that phrases consist of a particular grammatical relation—the function-argument relation (the relation between the verb *walking* and its object, such as the whole unit *on air*; or the relation between a preposition such as *on* and its object, such as *air*. As usual in artificial intelligence, finding the correct representation is 90 percent or more of the battle; it is the cornerstone of building theories. The search engine is secondary.

Turning now to the biological world, the same morals carry over. HMMs can find only linear patterns. Stochastic context-free grammars are far too broad a class of hierarchical patterns, so it is likely that search engines grounded on these will miss important transcription programs. What we need to understand first is the vocabulary of the transcription programs before we go looking for the programs themselves. It is unlikely that these insights will come from general inductive inference methods, except in an exploratory sense.

Neural Network Models

What about neural network (NN) approaches? Here too it is important to understand what work NNs can do. They cannot work magic. Today it is widely known that what they provide is function approximation: Given a set of data, NNs fit that data to a particular curve. For example, in the simplest case, it is known that a single-layer NN (one intermediate layer, one set of inputs, and one output) actually is carrying out classical principal components analysis. Conceptually, the picture is this: Given some cloud of data in, say, x , y , z space, where z is the dependent variable to be explained in terms of the variables x and y (we can think of z as the nucleotide sequence and x and y as factors that account for the observed sequence), then the network learning algorithm finds two things. First, it finds two *axes*—the principal components x' and y' —that optimally account for the dependent variable z . These components correspond to the NN “units” or “cells.” Second, the system finds the optimal *weights* to assign to each component to give the best fit to the data z . These correspond to the weights assigned to each NN cell or unit.

In this sense, NNs are doing statistical curve fitting. As statisticians know, one cannot build a good statistical model out of thin air: One has to know something about which variables might be related to which other variables. If one starts with a poor set of hypothesized variables x and y to explain z , then the NN search method cannot save us. For instance, if these are (obviously) poor descriptors such as say, the number of stop sequences, then no amount of NN learning can inform us adequately. In this sense, like HMMs, NNs can greatly help us explore a space of possible theories and can be extraordinarily efficient search engines for finding patterns in sequence data. Nonetheless, in the area of natural languages, NNs have not proved to be very useful except in the same places HMMs have been—for instance, in building systems that learn how to map text to speech. This is true, as it is for HMMs, because the topology of simple NNs best reflects the literally linear properties of a spoken sound sequence. Though there have been attempts to capture some of the hierarchical structure of human language via such networks (using recurrent [i.e., recursive or reentrant] nets), such attempts have been generally unsuccessful. If it is true that genetic transcription is far more sophisticated than natural language—as we show in the next section—then this result means that NNs will never give us the correct answers about transcription. What is needed is a new theory about the space of transcription programming language constructs.

In sum, NN learning algorithms can be efficient search engines for *existing* theories about linear language or nucleotide sequences and transcription, but their value for higher-order natural language or sequence constructs is more dubious. NNs can suggest possibly valuable new combinations of proposed theoretical variables, just as does principal components analysis does. However, NNs cannot invent new theoretical variables out of whole cloth. Once

again, starting with the correct representations, the right search spaces, is *the* most important factor.

THE SIMPLICITY OF NATURAL LANGUAGES AND THE COMPLEXITY OF GENETIC LANGUAGE

If natural languages and the language of the genes are not formal languages, then what are they? We have mentioned several times now that both nucleotide sequences and the transcription machinery itself seem more akin to a programming language than to natural languages, and that natural languages may be much simpler than the language of the genes. In this section, we show exactly how simple natural languages may be—specifically, that natural language syntax might be grounded on just a single, simple, computational combinatorial operation. Further, this operation, which seems central to all grammatical relations, does not seem to be directly reflected in the language of the genes.

Natural Grammatical Relations

Let us begin by defining what we mean by natural grammatical relations, the relationships that natural language syntax does seem to use. Surprisingly, there seem to be relatively few central relations (perhaps only four), defined over a local domain of binary branching tree structures. This constraint is interesting because, of course, given arbitrary tree structures—such as those available if we posited arbitrary hierarchical relationships—there could just as well be an *infinite* number of distinct grammatical relations. Yet most of these are not ever used in natural languages. For instance, we could well imagine that there is a relation between, say, the subject and the object of a sentence. Indeed, if we adopted an HMM or a context-free grammar model, there is nothing at all to block such a relationship. It is in this sense that HMMs and context-free grammars are too general and therefore cannot explain why natural languages are the way they are rather than some other way. Still worse, from the point of view of discovery procedures, is the fact that search algorithms that use only the space of possibilities defined by HMMs or context-free grammars use the wrong space.

What kind of space is right then? Here we can follow recent work of Epstein (1995). The basic natural grammatical relations are perhaps best exemplified by a simple picture, where *X* and *Y* denote *nodes* or entire subtrees or phrases, such as sentences, noun phrases, or prepositional phrases. We first note that the configurations are all binary branching (not a necessary property of tree structures generally).

Reviewing the configurations in figure 15.3, the first relation is essentially that of verb-object, or preposition-object (e.g., *ate ice cream* or *on the table*); more generally, this is the function-argument relation. The second relation is almost that of tree dominance, which is essential for hierarchical description,

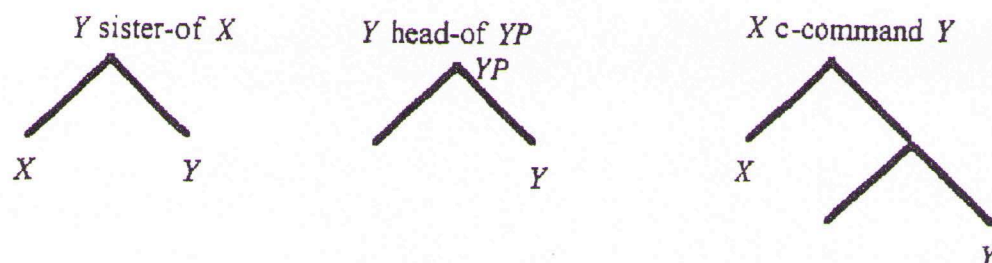


Figure 15.3 Three basic grammatical relations in natural language syntax.

as in a phrase such as *ate ice cream*, where the entire phrase is a verb phrase, denoted by the tree node *YP*, and *Y* is a subpart of the tree—in this case, *ice cream*. This is actually the notion “head-of”: Note that perhaps the most prominent property of a phrase—its type—depends on the feature propagated or inherited from the word that heads it up. For example, a verb phrase such as *ate ice cream* is built around a scaffolding that consists of first a *verb*: That is, we may think of the lexical property of the verb as being propagated up from *Y* to *YP* (= a verb phrase, or *VP*) in the figure 15.3. In this sense, this second relation defines the *kinds of phrases* that one can find in a language. The third relation may be more unfamiliar to nonlinguistic readers but is, in fact, one of the most important in natural language syntax: It is dubbed *constituent command*, or *c-command*: A node *X* c-commands a node (phrase) *Y* just in case the first branching node that dominates *X* also dominates *Y*. In our picture, *X* does c-command *Y* (because if we go up to the first branching node that dominates *X*, we find that this dominates *Y*), but *Y* does *not* c-command *X* (so the relation is not symmetrical). Intuitively, c-command is the notion of scope in natural language, similar to the notion of scope in logical calculi or programming languages: C-command defines the domain over which a variable can be bound. In natural languages, this corresponds to sentences such as “Whom did John think that Mary saw?” which can be rendered roughly as, “For which *z*, *z* a person, did John think that Mary saw *z*?” where the variable *z* is linked to *whom*. Note that if one drew out the syntactical structure for this sentence, we would have something akin to figure 15.4, wherein the variable *z* is c-commanded by *whom*. Because this kind of linking shows up again and again in modern linguistic theory as the foundation of what used to be called *transformations*, one can see that this configuration is an important one. These basic relations—function-argument, head-of, and c-command—seem to be the primitive building blocks for all other linguistic relationships.

Explaining the “Natural” Grammatical Relationships

We next show, following Epstein (1995), that these basic relationships all are accounted for by a single elementary computational operation based on the adjacent concatenation of tree structures. (The syntactical reflex of this idea was first proposed by Epstein [1995].) Note that this is a “natural” result in

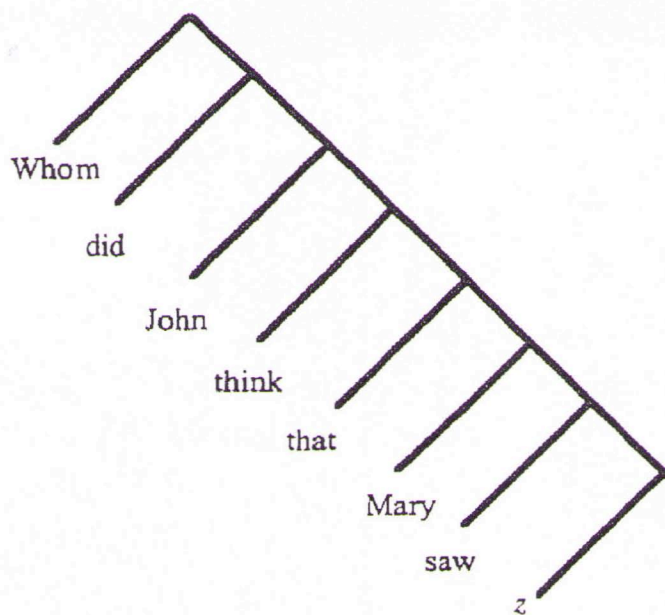


Figure 15.4 The c-command relation between *whom* and the (unpronounced) variable *z*, the object of *see*, is like the relationship between a quantifier and the variable it binds.

the sense that we know, on independent grounds, that human language syntactical structure is treelike (rather than purely linear, like beads on a string). Note also that if this result is correct, it automatically explains why HMM models based on linear concatenation do not do a very good job of accounting for human syntax. The central idea here is that hierarchical concatenation is the chief operation we need to derive natural language sentences. To show how this works, let us see how a sentence such as “John likes the ice cream” might be derived.

Hierarchical Concatenation

Let us first describe the concatenation operation itself. It is simply a *bottom-up tree composition*: We take two subtrees, *X* and *Y*, and “glue” them together, forming a new larger tree in a special way: Either the features of *X* or the features of *Y* are propagated to the new larger tree, forming a node of either type *XP* or *YP*. For example, suppose we have a verb *eat* (actually a subtree), and a subtree corresponding to *the ice cream* (a noun phrase, or NP). We combine these to form a VP. This abstract combination of *X* and *Y* as well as the specific example combining a verb and a noun phrase are shown in figure 15.5. The left half of the figure shows the two initial subtrees, drawn as triangles. The first triangle consists of the verb; the second consists of the noun phrase. The right half of the figure shows the result of the combinatory operation that glues these two triangles into a single larger one: (1) The features of the verb subtree are propagated up to a new node, the verb phrase (VP) node, that is the root or top of the new larger triangle, labeled *Z*; (2) we represent this top most point via a special set notation that marks *likes* as the “head” of this phrase; (3) the noun phrase subtree is pasted in place below.

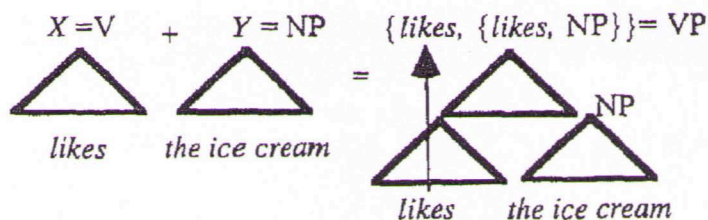


Figure 15.5 The basic operation of hierarchical (tree) concatenation. V, verb; NP, noun phrase; VP, verb phrase.

We dub this operation *bottom-up* because it pastes two smaller adjacent trees into a single larger one; it is *computational* in that we take this to be the operation of a parser proceeding from left to right through the sentence. In fact, this operation corresponds to one of the most common ways of parsing programming languages, so-called *LR parsing*, in which we paste together larger trees out of smaller ones, as shown in figure 15.5.

Deriving a Full Sentence

In this view, then, the derivation of a sentence proceeds by a sequence of hierarchical concatenation sets (what were called *derivation lines* in the original theory of Chomsky [1956]). In this case, the derivation steps are as follows, where by *form* we mean "construct a hierarchical structure like the triangle dominating *likes*":

1. Form the hierarchical (triangle, "tree") representation for *John*
2. Form the hierarchical representation for *the ice cream* (= Y in figure 15.5)
3. Form the hierarchical representation for *likes* (= X in figure 15.5)
4. Concatenate X and Y, forming an extended verb phrase, Z, corresponding to *likes the ice cream*
5. Concatenate Z with the triangle representation for *John*, forming a complete sentence, "John likes the ice cream"

In summary, note that thus far the whole sentence is derived by a sequence of hierarchical concatenation steps, and only these.

Deriving Grammatical Relations

The important point now is to show that if we assume this operation of tree concatenation to be the basic primitive of syntax, then it follows that the only grammatical relations we see will be precisely those described earlier. In this sense, we may say that natural language syntax uses only a single operation of hierarchical, adjacent tree concatenation.

Let us see why *these* basic relations follow and no others. The central insight is that two elements may be *related* in the grammar if and only if they are adjacent or visible to each other at the time of tree concatenation: that is, at the derivation step that glues the two trees together. What *visible* means is this: Let us say that a tree such as the noun phrase *the guys*, ordinarily

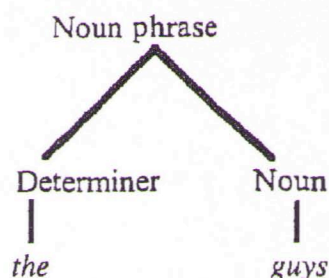


Figure 15.6 A conventional tree representation for the noun phrase *the guys*.

represented as in figure 15.6, is represented by the following set of *terms* (following Epstein, 1995):

Noun phrase = {Determiner–*the*, {Determiner–*the*, Noun–*guys*}}

Figure 15.6 shows that the tree or noun phrase corresponding to *the guys* was built out of the composition operation that pasted together *the* and *guys*, forming a new tree with a new root (topmost) node. Initially, *the* was simply the set (subtree) {Determiner–*the*}, where we have tacked on the syntactical category *Determiner* for ease of reading. Similarly, *guys* was the set (subtree) {Noun–*guys*}. The concatenation operation is as described previously, and we select one of the two combines as the name of the new root tree. We now propose simply that *two syntactical elements (i.e., hierarchical structures, trees) can enter into a grammatical relation if and only if there is some point in a derivation (sequence of concatenations) at which both trees at which both trees are made connected terms (members) of the same common subtree (via the concatenation operation).*

For example, in our figure, note that *the* and *guys* can enter into a common grammatical relation because they are both terms of some other set at the time of their concatenation, and they are directly related via concatenate—viz., the set that represents the noun phrase. This relation is, of course, simply the *sister-of* relation described in figure 15.3 (and also described as the *function-argument* relation).

The same property holds for the other two basic grammatical relations mentioned. For the head-of relation, note that in the mother tree {{*the*, {Determiner–*the*, Noun–*guys*}}, the first term in the set, *the*, can, by definition, be related to either of the other two terms: In other words, the root node can be related to either of its immediate daughters. However, this is just the head-of relation, as in figure 15.3. For c-command, note that *X* and *Y* are hierarchically concatenated; then *X* c-commands all the elements (terms) of *Y*. For instance, if *X* = the noun phrase tree corresponding to *the guys*, then *X* = {*the*, {Determiner–*the*, Noun–*guys*}}. If *Y* = the verb phrase tree corresponding to *like the ice cream*, then *Y* = the five-term set {*like* {Determiner–*the*, {*the*, Noun–*ice cream*}}}, and *X* c-commands every one of those terms (and not vice versa, crucially). In contrast, *ice cream* cannot c-command *guys* because, at the time the noun phrase corresponding to *the ice-cream* was built

(concatenated out of two parts), *guys* was not part of that set of terms. In this way, the asymmetrical nature of c-command is derived.

In contrast, certain relations can *never* obtain: For example, because *the ice-cream* and *the guys* are never concatenated together directly but only after trees above them have been built, there can be no grammatical relation that holds between subjects and objects, as we find to be the case in natural languages. Note that there is no *logical* reason this should be so otherwise, unless we assume some fundamental constraint such as the concatenation operation. In other words, in a general context-free system, we can easily write a grammar that relates subjects and objects. Why we do not find any such relations remains a mystery, unless there is a more fundamental constraint that underlies natural languages. As we have sketched, this law seems to be a simple one: Natural languages are formed by a single algebraic operation of hierarchical concatenation. This, then, is *natural language*.

CONCLUSION: NATURAL AND GENETIC GRAMMARS

Plainly, the concatenation operation is simple. It is *adjacency* as extended from strings to trees. Just as plainly, the proposed "grammars" for genetic transcription are vastly more complicated. There appears to be nothing in natural languages corresponding to splicing followed by distinct reading frames. To take another example, Searls (1993) uses the logic programming language Prolog to describe exons and translated regions, which is fine except that with Prolog we also can describe a connection between subjects and objects in natural languages that we do not see. Of course, all this says is that the constraint lies in the particulars of the program that the scientist writes rather than in the constraints of the programming language itself. If this is so, it is left to the programmer or scientist to discover the constraints; the space of possible theoretical descriptions given by the representation language—in this case, Prolog—is vast. If this is so, then we still do not have any strong insight into what constitutes the language of the genes. We know what it is not: It is *not* a finite-state language or a context-free language, but neither is it a context-sensitive language. If anything, the language of the genes is much more like a programming language whose constraints we do not know (or whose programs we do not know). Now, the problem of identifying a program's details from observations of its input and output behavior is very, very difficult; even in the case of finite-state programs, the problem is unsolvable unless we assume that we know other constraints (such as the number of states in the program). Yet this is the task that molecular biologists seemingly have set for themselves. Given that we currently have no general means of carrying out such inductions, it would seem best to work out particular case studies of reverse engineering—looking at transcription in the way that Collado-Vides (chapter 9) has done and then determining what kind of computer program is best suited for describing the engineered constraints

we do observe. Our knowledge here seems just at the starting point, so much so that we must amass many more case studies before we can come up with the generalizations that will tell us what the genetic constraints are. By way of comparison, it has taken more than 40 years to determine that, in the end, syntactical relationships in natural languages are, in fact, derivable from a single, simple algebraic operation. We can discover what language the genes are speaking to us only by more years of careful listening.