The Linguistics of DNA

In the quest to understand the language of life, a promising strategy is to construct a grammar of genes

David B. Searls

Linguistic metaphors have been a part of molecular biology ever since the structure of DNA was solved in 1953. Biologists speak of the genetic *code*, of gene *expression* and of *reading frames* in nucleic acids. DNA is *transcribed* into RNA, which is then *translated* into protein. Certain enzymes are even said to *edit* RNA. Despite all this linguistic terminology, however, there has been little effort to apply the tools of formal language theory to the problems of interpreting biological sequences.

At about the same time that Watson and Crick were laying the foundation of molecular biology with their model of the double helix, Noam Chomsky was initiating an equally fundamental revolution in linguistics. The engine of this revolution was the idea of a generative grammar: a set of rules that can generate all the grammatically acceptable sentences of a language, without generating any erroneous ones. Through this device a relatively small set of rules can capture the structure of a potentially infinite language. Chomsky's primary interest was in understanding natural human languages such as English, but his work was immediately successful in elucidating the more abstract and formal languages of mathematics and computer science. It now appears that the idea of a generative grammar may also be a powerful tool for expressing the biological messages inscribed in DNA and RNA.

Finding effective methods for reading the language of nucleic acids is rapidly becoming an issue of practical concern. The Human Genome Project, in its most ambitious phase, proposes to record the entire sequence of the three billion DNA base pairs that make up the human genetic endowment. If this archive is to be of any use, biologists will have to be able to identify and retrieve meaningful segments of it. This is no trivial challenge, given that functional genes make up only a few percent of the total DNA, and our knowledge of how they are controlled by surrounding elements of the genome is still rudimentary. Equipped with knowledge of the linguistic structure of the genome, one can endeavor to write a computer program that parses genes and other high-level features of DNA.

David B. Searls is research associate professor of genetics at the University of Pennsylvania School of Medicine, with a secondary appointment in computer and information science. He received bachelor's degrees in philosophy and life sciences from the Massachusetts Institute of Technology, and a Ph.D. in biology from the Johns Hopkins University. Subsequently, he turned his attention to computational biology and received a master's degree in computer science from the University of Pennsylvania. He is currently co-director of the Computational Biology and Informatics Laboratory there. Address: Department of Genetics, Room 475, Clinical Research Building, University of Pennsylvania School of Medicine, 422 Curie Blvd., Philadelphia, PA 19104-6145.

Grammars and Languages

A generative grammar is more than a set of rules and recommendations for the correct use of language; it is a mechanism for actually *producing* all the syntactically correct utterances in a language, and only those utterances. Here is a generative grammar for a tiny subset of English:

sentence → *noun-phrase verb-phrase*
noun-phrase → *article modified-noun*
verb-phrase → *verb noun-phrase*
modified-noun → *noun* | *adjective modified-noun*
noun → `linguist` | `biologist`
verb → `sees` | `believes`
adjective → `young` | `famous`
article → `a` | `the`

The grammar consists of "production rules," which specify the structure of an English sentence in progressively greater detail, until finally individual words are introduced. The first rule states that a sentence consists of a noun phrase followed by a verb phrase; the second rule then defines a noun phrase as an article followed by a modified noun; the third rule declares that a verb phrase is made up of a verb and a noun phrase. The fourth rule offers a choice: The vertical bar | simply means "or," so that a modified noun may be either a noun by itself or an adjective followed by a modified noun. The remaining production rules give a small vocabulary of nouns, verbs, adjectives and articles. All of the italicized words are "nonterminal" symbols, meaning that they never appear in the actual sentences that are the final output of the grammar; lexical items such as `linguist` and `sees`, on the other hand, are "terminal" symbols that will be part of the generated sentences. The arrow within each rule can be read as "produces" or "expands into."

To generate a sentence, all that is needed is to apply the rules, one after another. The process begins with the top-level rule, in which the single symbol *sentence* is replaced by the sequence of symbols *noun-phrase verb-phrase*. Then more rules are applied to expand these symbols in turn, so that *noun-phrase* yields *article modified-noun*, and so on. Eventually all of the nonterminals are replaced by terminal symbols. An entire sequence of rule applications, called a derivation, is shown in Figure 2. It leads to the sentence: "The linguist sees a famous young biologist."

With the grammar given here, which generates a very limited variety of forms, it is easy to verify that all of the sentences are grammatical (if slightly inane). Despite this simplicity, the grammar could in theory produce an infinite number of sentences. The source of its

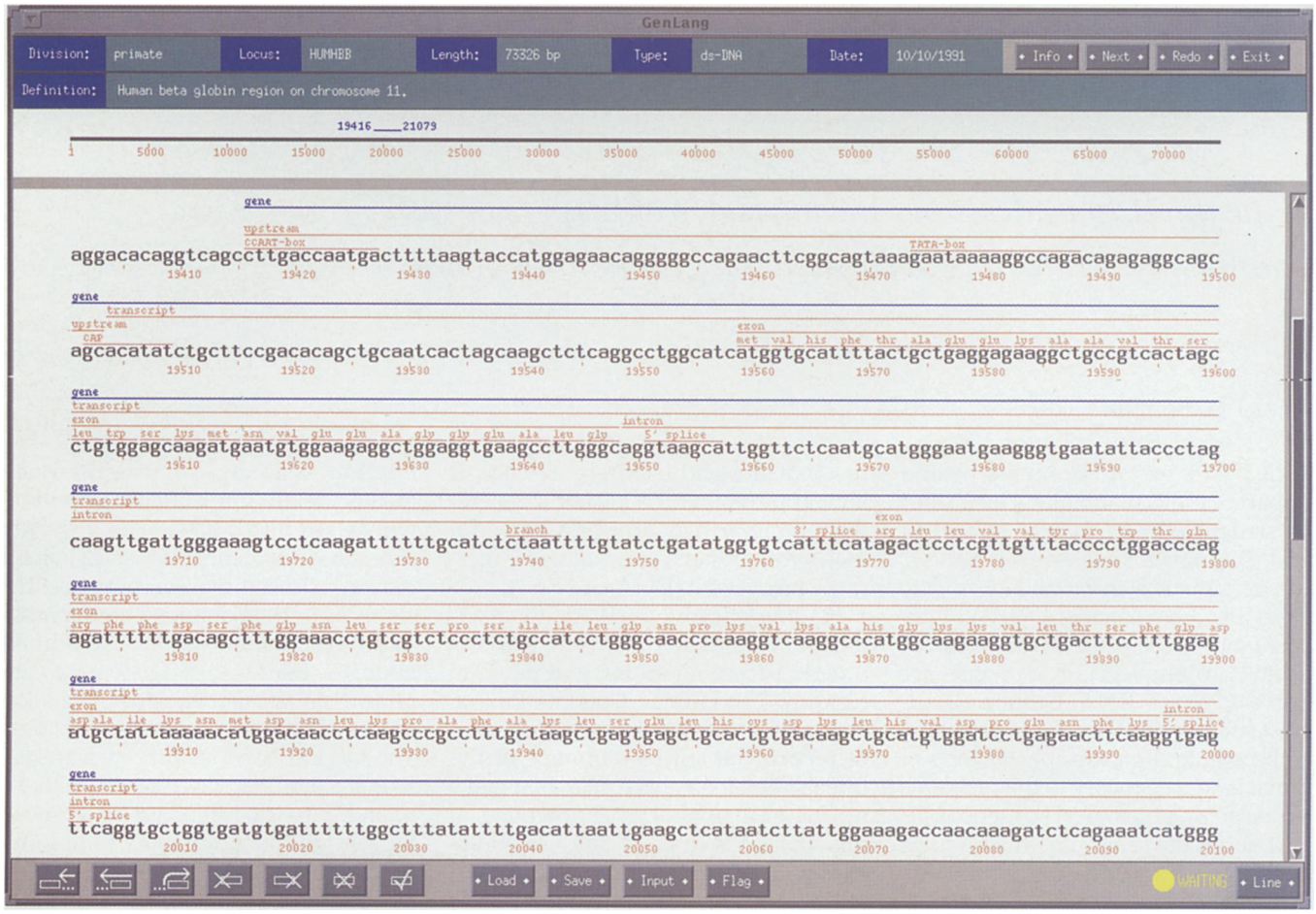

Figure 1. Structure of a gene is revealed by linguistic analysis. In the author's laboratory, grammars are used to parse DNA sequences, elucidating the hierarchical organization of genes and other high-level components of the genome. Shown here is the first half of a human hemoglobin gene, one of five similar genes distributed in a region of DNA some 73,000 nucleotides in length. The long list of letters is the nucleotide sequence; the variously colored lines represent features recognized, including the gene itself. Although a simple syntactic analysis cannot unambiguously identify genes, grammars provide a promising formal framework for exploring higher-order structure in biological sequences.

infinite capacity is the rule for *modified-noun*, which has a recursive structure: It invokes itself. Each time *modified-noun* expands into the sequence *adjective modified-noun*, there is another instance of *modified-noun* to be expanded. The process can string together an unlimited number of adjectives. Although the sentences would quickly grow tiresome and even nonsensical, there is no point at which adding just one more adjective makes the sentence ungrammatical.

Such a toy grammar can hardly begin to express the diversity of English syntax. To make the grammar more realistic, it would have to be greatly elaborated, but that exercise would probably reveal more about the peculiarities of natural languages than about the nature of grammars. A better strategy is to look instead at the grammars of still simpler languages, namely the abstract and formal languages studied in mathematics and computer science—and molecular genetics.

A Hierarchy of Languages

Formal languages look nothing like the languages of everyday experience. The theory of formal languages defines a language as a set of "strings" formed by writing down a sequence of symbols drawn from some specified alphabet. The strings do not have to mean anything; indeed, the term "string" is favored over more familiar alternatives such as "word" or "sentence" because it carries no connotation that the sequence of symbols should be meaningful. A string is a member of a language if it satisfies some formally defined property, such as being derivable from a given grammar—there are no other criteria.

As an example, consider the alphabet made up of the two symbols 0 and 1, and the language consisting of all possible strings drawn from this alphabet. The language, which is equivalent to the set of all integers expressed in binary notation (allowing leading 0's), includes strings such as 0, 101, 111011, 010101101, and countless others. Indeed, the language is infinite, even though the alphabet has only a finite number of symbols.

The language of binary integers has a simple generative grammar:

$$S \rightarrow 0S \mid 1S \mid \varepsilon$$

Here the symbol S is a nonterminal, whereas 0 and 1 are terminals. As in the English grammar, the vertical bar means "or." The one new element of the grammar is the symbol ε (the Greek letter epsilon), which represents the "empty string," the string of zero length. The three alternative productions can be translated into words as follows: "Any instance of the nonterminal S yields the string $0S$ or the string $1S$ or the empty string." Choosing the third possibility, ε, amounts to erasing the S.

The operation of the grammar is straightforward. Beginning with S, any of the productions can be chosen. If the first production selected is $S \rightarrow 1S$, then

the effect is to replace the solitary S with the string $1S$. This derivation is represented as $S \Rightarrow 1S$, where the double arrow signifies that this is an application of a grammar rule, not a statement of the rule itself. The S in the new string $1S$ is now subject to replacement in turn; if the production $S \rightarrow 0S$ is chosen this time, the derivation is $1S \Rightarrow 10S$. Now the rule $S \rightarrow 1S$ might be applied again, yielding $101S$. Finally one selects the third alternative production, $S \rightarrow \varepsilon$, and the S disappears, leaving the string 101. Since this string has no nonterminal symbols, it cannot be transformed further. The process of building the string is summarized as follows:

$$S \Rightarrow 1S \Rightarrow 10S \Rightarrow 101S \Rightarrow 101$$

The grammar can generate strings of any length, and either 0 or 1 can appear at any position within a string, so that all binary integers are included in the language.

Introducing more restrictive production rules gives rise to languages with a more complicated structure. Consider this grammar:

$$S \rightarrow 0S \mid 1T \mid \varepsilon$$
$$T \rightarrow 0T \mid 1S$$

Here S and T are both nonterminals; the alphabet of terminals again consists of 0 and 1. What set of strings is generated by the grammar? It is the language of all binary strings having an even number of 1's, as in this sample derivation:

$$S \Rightarrow 0S \Rightarrow 01T$$
$$\Rightarrow 011S \Rightarrow 0110S \Rightarrow 0110$$

Palindromes and Repeats

The grammars of binary numerals presented above have a distinctive form. In every rule the left-hand side consists of a single nonterminal symbol, and the corresponding productions on the right-hand side either have no nonterminals (as in $S \rightarrow \varepsilon$) or else they have a single nonterminal as their final symbol (as in $S \rightarrow 0T$). It follows that strings generated by these rules can grow only from one end. A grammar that fits this description is called a regular grammar, and the corresponding language is a regular language. (The same languages can be specified in other ways as well; a common technique is the formalism known as regular expressions, often used in searching for patterns in text.)

Regular languages have a number of attractive properties, most notably the computational efficiency of recognizing them. But they also have limitations. A regular language can capture the notion "all strings of 0's and 1's," or "all strings with an even number of 1's," but it cannot represent certain other concepts. Of particular importance in a biological context, no regular language can specify the set of all palindromic strings—that is, strings that read the same forward and backward. There are many proofs of this assertion, but I shall merely offer an informal argument in support of it: The way to construct a palindrome is to build it from the middle outward or from the two ends toward the middle, but the defining trait of regular grammars is that they add symbols only to one end of a string.

To write a grammar for palindromes, some of the restrictions on the form of the production rules must be relaxed. Specifically, nonterminals must be allowed in any position on the right-hand side of a rule. Here is a simple grammar that produces palindromes drawn from the alphabet consisting of 0 and 1:

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$$

Note that the nonterminal S appears in the middle of the productions $0S0$ and $1S1$, which would not be allowed in a regular grammar. The grammar is guaranteed to generate only palindromes because the string grows outward from the center, with each production adding a balanced pair of 0's or 1's. A sample derivation would look like this:

$$S \Rightarrow 0S0 \Rightarrow 01S10$$
$$\Rightarrow 011S110 \Rightarrow 011110$$

All the palindromes generated by this grammar have an even number of symbols, but a grammar that can accommodate odd-length palindromes, such as 01010, is only a bit more complicated.

Grammars with no restriction on the number or placement of nonterminals on the right-hand side of a rule are called context-free grammars, and the associated languages are context-free languages. Chomsky showed that context-free languages are more powerful than regular languages, precisely because they can express concepts such as that of a palindrome. Moreover, the class of context-free languages includes the class of regular languages, since a regular grammar is simply a special case of a context-free grammar.

Context-free languages are not at the

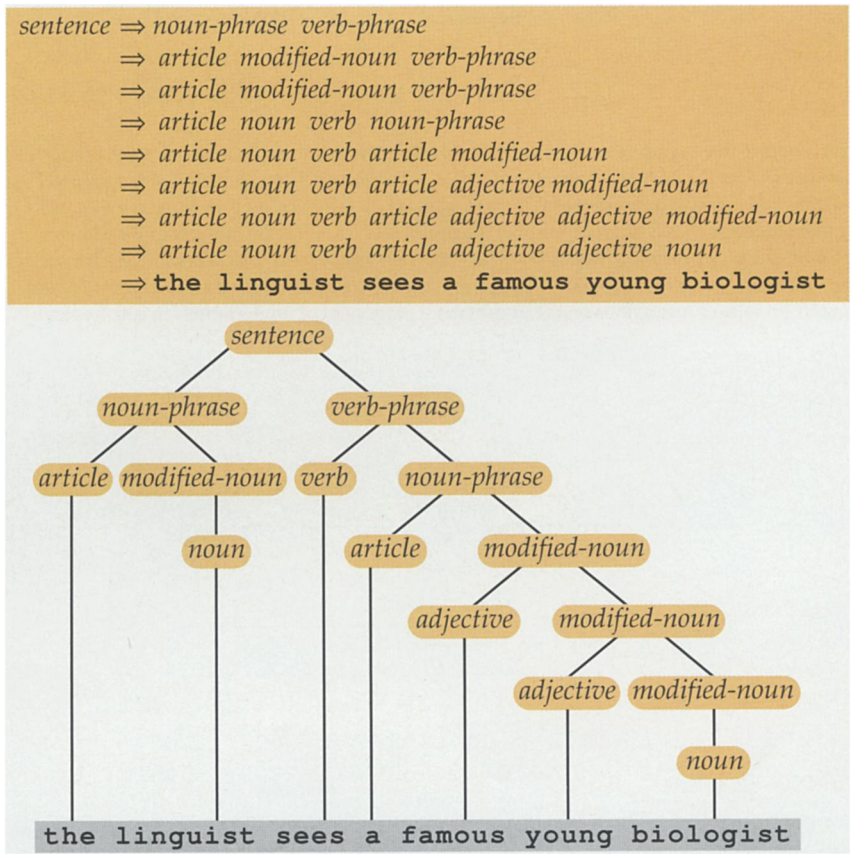

Figure 2. Parse tree represents the syntactic structure of an English sentence. The sentence is generated according to the rules of a grammar in the series of steps shown at the top. All the stages of this derivation can then be arranged in a tree structure (which by convention is drawn inverted, with the root at the top). Formal languages, as well as the languages of biological sequences, are also well suited to this kind of analysis.

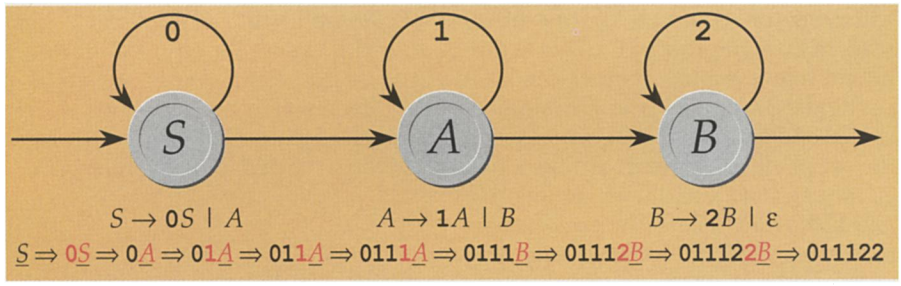

Figure 3. Regular languages, the simplest languages in the hierarchy defined by Noam Chomsky, are described by regular grammars and by the abstract machines called finite-state automata. An FSA is a collection of states *(circles)* and transitions *(arrows)*. It generates language by emitting symbols associated with some of the transitions. The FSA shown here begins in state *S*, where it can emit any number of 0's by making transitions back to the same state; on moving to state *A* it can generate 1's in a similar way; then in state *B* its output consists of 2's. The grammar below the diagram specifies the same language, namely all strings made up of any number of 0's followed by any number of 1's followed by any number of 2's. Each grammar rule shows how a nonterminal symbol on the left-hand side of the arrow may be replaced by a rule body on the right-hand side; the vertical bar means "or"; the ε rule simply erases the nonterminal. All rule bodies have at most one nonterminal, and it is the rightmost symbol. The derivation below the grammar shows how a string develops by repeated application of the grammar rules. At each step the underlined nonterminal is replaced by the rule body shown in color in the following step.

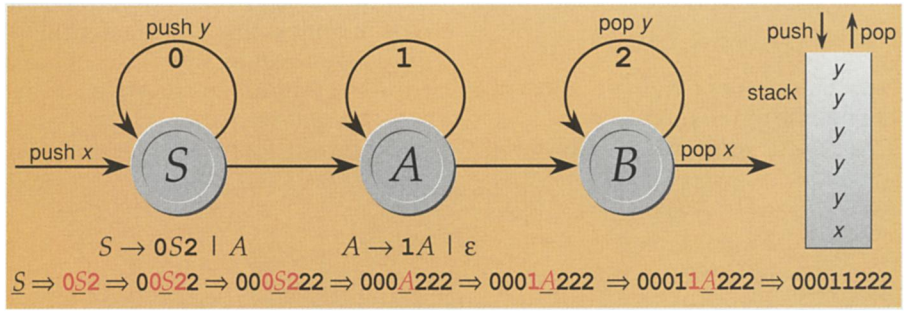

Figure 4. Context-free languages are generated by pushdown automata, in which an FSA is augmented by a simple memory mechanism—a "stack" where symbols are"pushed" and "popped" during state transitions. The automaton shown generates 0's followed by 1's followed by 2's, as in the regular language above, but requires that the number of 0's equal the number of 2's. The machine counts the 0's by pushing a *y* onto the stack for each 0 generated, and later emits a 2 for each *y* it pops; the string ends when the machine pops an *x* that was pushed at the outset. The equivalent grammar form relaxes restrictions on the number and placement of nonterminals in rule bodies.

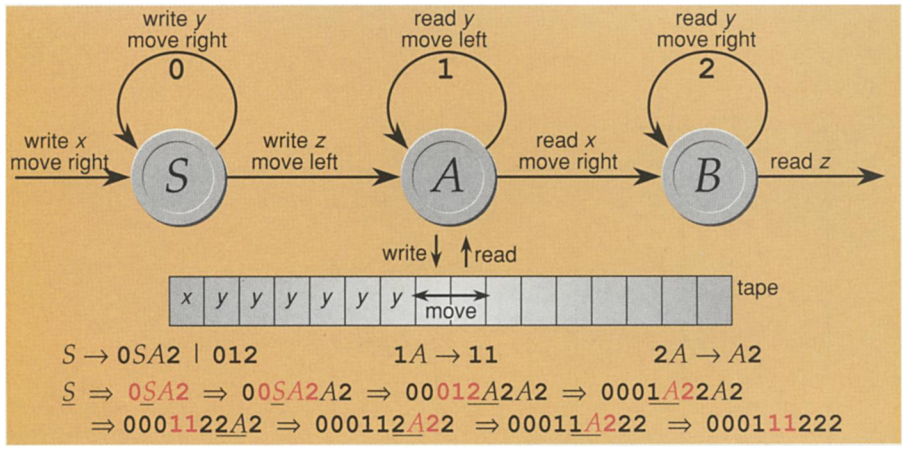

Figure 5. Context-sensitive languages require a machine with a more versatile memory—a tape on which the automaton can read or write symbols, and then move left or right. This machine can produce all strings with equal numbers of 0's, 1's, and 2's, in that order. The grammar has rules with more than one symbol on the left-hand side, which serve, for example, to transpose symbols. In a context-sensitive language the length of the tape is proportional to the length of the string generated, and the left-hand side of a rule is never longer than the right-hand side. Without these constraints, the language is called recursively enumerable, and the automaton is a Turing machine.

pinnacle of Chomsky's pyramid of languages. A context-free grammar can specify palindromes and numerous other interesting patterns, but there are also many concepts that escape the context-free formalism. A *copy language*, for example, consists of all strings in which the first half of the string is identical to the second half. Where a palindrome might read `011110`, a member of the copy language would read `011011`. Generating such repeats may seem little different from generating palindromes, but in fact the task is much more difficult to accomplish.

Creating a grammar for a copy language entails a further loosening of the constraints on the form of the production rules. This time the requirement to be jettisoned is the one stating that the left-hand side of each rule must consist of a single nonterminal symbol. By allowing additional symbols on the left-hand side, one can indeed create a grammar for the copy language, although it is too complex to be presented here. The key to its operation is that the presence of multiple symbols on the left-hand side of a rule allows for arbitrary movement of symbols to the left and right in the developing string, so that (for example) a palindromic string can be systematically rearranged into a string from a copy language.

A copy language is an example of a *context-sensitive* language. In a context-sensitive grammar the one constraint remaining on the form of a rule is that the right-hand side be at least as long as the left-hand side. Beyond the context-sensitive languages there are languages that can be specified only by dropping all constraints on the form of a grammar rule. This rarefied class of languages specified only by unrestricted grammars, however, is not easy to characterize; the examples tend to be rather contrived and mathematical.

The Chomsky Hierarchy

The ranking of grammars and languages in Chomsky's classification is sometimes confusing. Going up the ladder from regular through context-free and context-sensitive to unrestricted languages, the grammars are subject to fewer constraints, but the languages themselves are more narrowly defined. A context-free grammar can include rules that would be forbidden in a regular grammar, but a context-free language is more powerful not because of what it includes but because of what it excludes—such as strings that are not

palindromes. Ascending the hierarchy allows one to be more specific, and thus to describe a wider variety of languages.

One way of understanding the Chomsky hierarchy is by considering the patterns inherent in the languages. A regular grammar can generate overall patterns in strings, such as alternating 0's and 1's. What a regular grammar cannot do is maintain dependencies at arbitrary distances within a string of symbols; it cannot use a symbol in one part of the string to decide what symbol to write in another part. Context-free languages allow certain dependencies within a string, for example where the first symbol is matched with the last symbol, the second with the penultimate, and so on. Any pattern of dependencies that can be drawn so as not to cross one another (as in a palindrome) is said to be nested, and nested dependencies are characteristic of context-free languages. When dependencies cross (as in a copy language), a context-sensitive language is mandated, because crossing dependencies can be established only with the freedom of movement that nonterminals enjoy during a context-sensitive derivation.

Another approach to understanding the hierarchy looks at the kind of machine needed to generate each class of language. The machines in question are not devices of steel or silicon but abstract and idealized computing mechanisms. For a regular language, the machine required is a finite-state automaton, or FSA. As the name suggests, such a machine has a finite number of states—typically one state for each nonterminal in the grammar—which can be represented as nodes connected by arrows showing possible transitions between the states. The FSA begins operation in the state corresponding to the start symbol, then with each symbol produced moves along an arrow to a new state. An important property of an FSA is that it has no storage facility apart from the collection of states; it has no way of recording information for later use, and so it can produce patterns only when they are "hard-wired" into the connections between states.

More powerful languages are associated with more sophisticated machines. A context-free language is generated by a "pushdown automaton," which consists of an FSA augmented by a "stack" that provides a limited form of auxiliary memory. The stack works like a stack of cafeteria trays in that only the topmost item is accessible; it is also known as a last-in, first-out buffer. The stack can be used to generate palindromes by pushing a copy of each symbol onto the stack as it is produced, then popping the symbols off the stack in reverse sequence.

For a context-sensitive language the appropriate machine is one with a more versatile memory—one based on a tape that can move in either direction. The machine can read from or write to the tape and move left or right on it, which allows for the erasures and rearrangements required in dealing with copy languages and other manifestations of crossing dependencies. The one constraint is that the length of the tape be proportional to the length of the generated string, so that the space used for intermediate calculations can never exceed some fixed multiple of the string length.

When the memory tape is allowed to grow to any length, the resulting automaton is a Turing machine, which is the device needed to generate an unrestricted language. The Turing machine is a kind of universal computer; no one has yet found any enhancement to this computing architecture that would allow it to execute some algorithm that it cannot already handle. Thus any language that can be produced by a digital computer program can also be precisely specified by some unrestricted grammar, and vice versa.

When viewing languages as sets of strings, it is natural to ask what happens when languages are combined by the operations of set union and set intersection. The union of two languages is a concept that should be familiar to anyone who is bilingual, and who therefore recognizes strings in either of two languages. Two people who speak closely related languages might define the intersection of their languages as the set of strings they both recognize. Mathematicians investigate the "closure" of collections of sets under such operations. A collection of languages is closed under the operation of set union if the union of any two languages in the collection is also a language in the same collection. All the levels of the Chomsky hierarchy are closed under set union; in other words, the union of two context-free languages (for example) must still be a context-free language. It is interesting, however, that the intersection of two context-free languages—the strings those languages have in common—cannot be guaranteed to be context-free. This fact may have some bearing on the linguistic status of DNA.

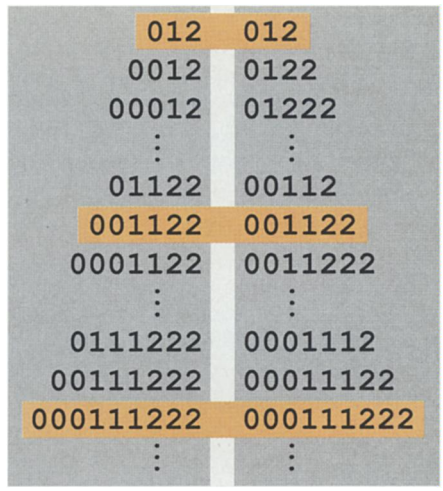

Figure 6. Union and intersection of languages form new languages that may have different properties. Here two languages are shown side by side, both based on sequences of 0's, 1's and 2's. In the language on the left, the number of 1's in each string must equal the number of 2's, whereas in the language on the right the number of 0's always equals the number of 1's. Both languages are context-free, and their union (the set of all strings that are members of either language) is also context-free. However, the intersection of the languages (the set of strings common to both) is the language with equal numbers of 0's, 1's and 2's, which is greater than context-free.

Although the term *generative* grammar suggests only the production of language, grammars are equally useful in recognizing and analyzing language; they can listen as well as speak. One way to do this is to apply the rules of the grammar "in reverse," starting with a string of terminal symbols and build-

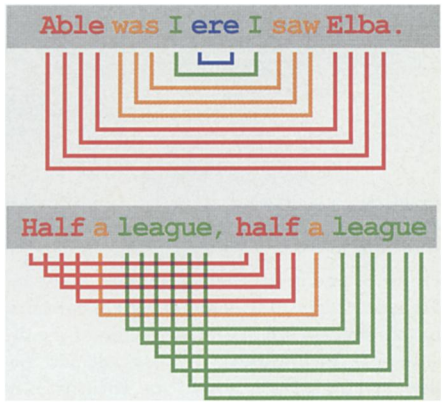

Figure 7. Dependencies between distant elements of a string characterize the nonregular languages. If the dependencies are strictly nested, as in the palindrome at the top, the language is typically context-free. When dependencies of unrestricted extent cross one another, as in the example of a copy language at the bottom, the language is beyond context-free.

ing up progressively more abstract categories. A program that takes as input a grammar and a string and decides whether the string is a member of the corresponding language is a recognizer—a more practical version of the abstract machines described above. In the case of English, a recognizer accepts a string of words as a grammatical sentence if the words can be organized into lexical categories, such as *noun* and *verb*, then into phrases, then into still higher-level structures, until finally the entire string is subsumed under the grammar's start symbol, *sentence*. The time it may take to process a string with a general-purpose recognizer for a class of languages increases sharply as one ascends the Chomsky hierarchy—a price paid for greater expressive power.

With a little more effort, a recognizer can be extended to serve as a parser, which not only passes judgment on a string but also produces an analysis of its syntactic structure. The result is usually displayed as a parse tree, a downward-branching diagram that reveals the hierarchy of grammatical categories implicit in the derivation. Parse trees can also be produced in the process of generation to represent what Chomsky, in the context of natural language, called the deep structure of a sentence. Chomsky's subsequent work has been much concerned with transformations, or rearrangements, of parse trees. Transformations can systematically produce wholesale changes in the surface structure or word order of a sentence, reflecting different ways of expressing essentially identical syntactic constructs, such as "The linguist sees the biologist" and "The biologist is seen by the linguist."

A further extension of a recognizer allows it to produce output as it reads input. For example, in the course of changing states, an FSA might read a symbol and at the same time emit a symbol from an entirely different alphabet. A machine of this kind is a transducer; it can travel along one string and generate another string that is related to the first by rules inherent in the construction of the machine. As we shall see, the living cell has a number of transducers, as well as analogues of linguistic transformations, dependencies and recognition.

The Genetic Code

How do these linguistic notions apply to DNA and other biological macromolecules? At first glance, DNA, RNA and proteins are all linear, one-dimensional molecules, made up of subunits strung together like the links of a chain. In the case of DNA, the links are the nucleotide bases adenine, cytosine, guanine and thymine, abbreviated `a`, `c`, `g` and `t`. At the most superficial level, DNA can be viewed as the language made up of all strings drawn from this alphabet; all possible strings are members of the language because it appears that no physical or chemical constraints would prevent some particular sequence or class of sequences from being assembled. But viewing DNA as such a totally unconstrained and unstructured language ignores both its role as a genetic archive and its status as a participant in the biochemical processes of a living cell. Although it may be true that any base sequence could be constructed in principle, only a very small subset of possible sequences appear in the living world. Grammars might help to characterize that subset.

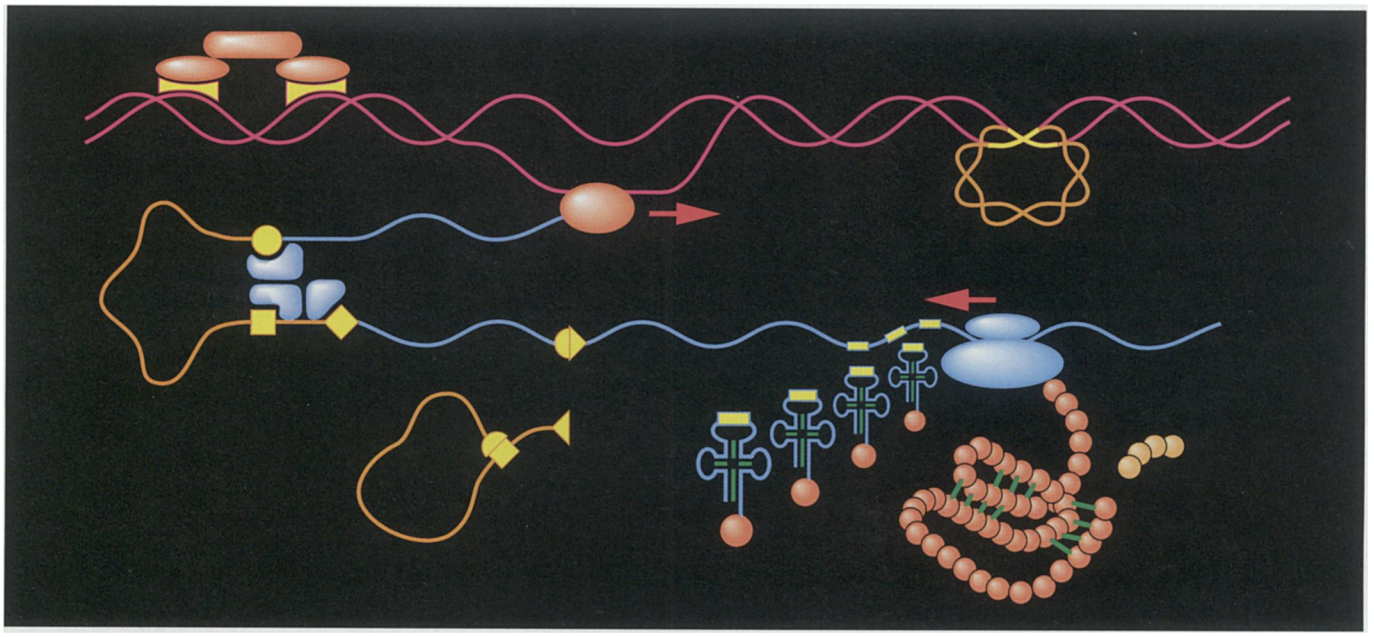

Figure 8. Linguistic themes abound in the mechanism of gene expression, illustrated here in a highly compressed and schematic way. Transduction, or the element-by-element mapping of one string to another having a different alphabet, is marked by red arrows; instances of this process are the transcription of the double-helical DNA *(purple)* into RNA *(blue)* by the enzyme RNA polymerase, shown as an ellipsoid near the middle of the diagram, and the subsequent translation of RNA into protein *(orange)* by the snowman-shaped ribosome at the lower right. Recognition of specific classes of strings is indicated by yellow, as at the upper left where regulatory proteins find promoter sequences in the DNA that help determine when and where transcription should begin. Transformation, the piecewise or wholesale modification of a string, is indicated by orange sequences, such as the intron being spliced out of the RNA at left. The splicing is mediated RNA-protein complexes that recognize splice signals shown as yellow circles, squares and diamonds; the resulting lariat-shaped piece of RNA is discarded. The ribosome, another RNA-protein complex, is a transducer from triplets of RNA nucleotide bases to amino acids, shown as red spheres. The cruciform shapes are transfer RNAs that act as triplet recognizers (at their yellow end) and carry the corresponding amino acids for the ribosome to add to the protein. Dependencies between segments of a string are shown in green; they are most obviously manifested in the folding of protein and RNA. Part of the newly synthesized protein is shown being clipped off in a process called post-translational modification, which is another example of transformation. Even DNA often undergoes transformation, for example via the insertion of transposable elements such as the small circular DNA molecule at the upper right, which is shown recognizing a specific site.

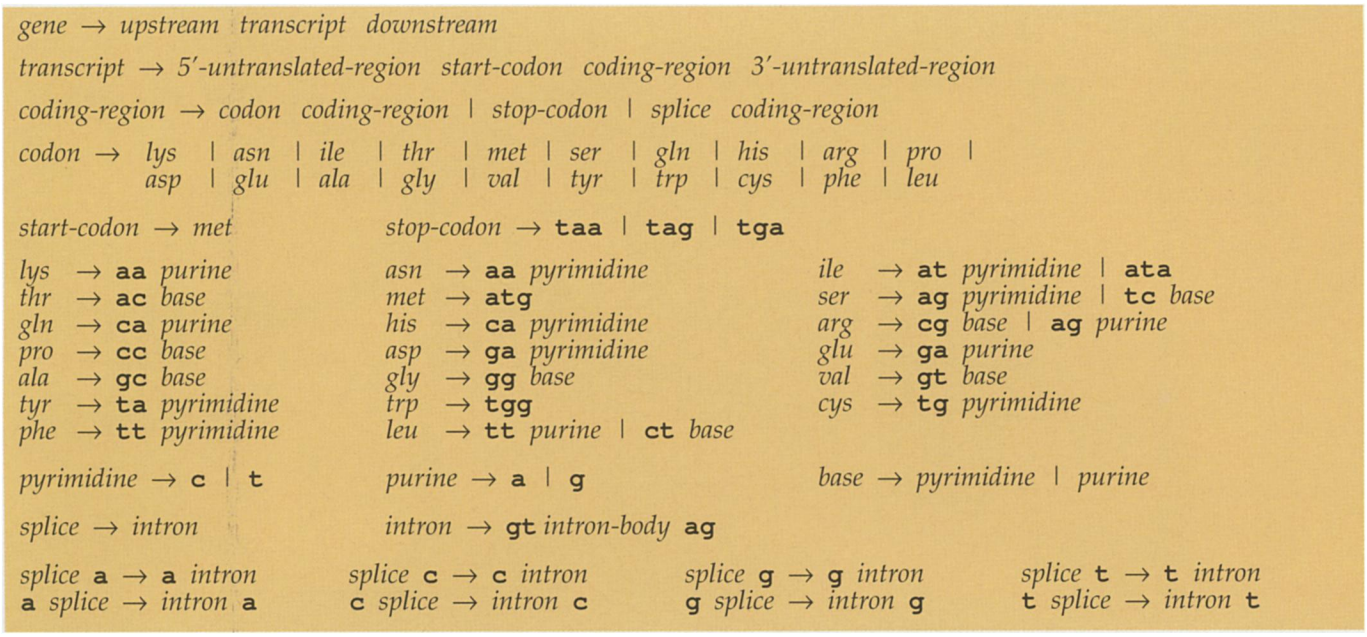

Figure 9. Partial grammar specifies some of the structure of a typical protein-encoding gene. The *transcript*, which is the part of the gene that is copied into messenger RNA, has flanking *5′-* and *3′-untranslated-regions*, around a *coding-region* initiated by a *start-codon*. The *coding-region* consists of a *codon* followed by more *coding-region*, a recursion that ultimately terminates in a *stop-codon*. A *stop-codon* is any of the three triplets indicated, whereas a *codon* is any of the 61 other possibilities, given in individual codon rules. Some of those rules refer to the nucleotide classes *purine* and *pyrimidine* to capture variability in the third codon position. The rule for *coding-region* also allows for a *splice* at any point in the recursion, resulting in the insertion of an *intron*. A series of context-sensitive *splice* rules allow introns to shift left or right, reflecting the fact that introns can appear within as well as between codons. An *intron* of this type is generally bracketed by splice signals including gt and ag. This simple syntactic description omits many other imperfectly understood signals, such as promoters of transcription usually found in the region *upstream* of the *transcript*.

Before proceeding further with a linguistic analysis of biological sequences, it would be well to briefly review the actual structure and chemistry of nucleic acids and proteins. The architecture of DNA is well known: Two strands of nucleotides, each thousands or millions of bases long, twine around each other in a double helix. The two strands are held together by a specific pattern of hydrogen bonding in which g mates with c and t with a. Thus the strands have complementary sequences—wherever a g appears in one strand there must be a c in the other strand, and likewise every t requires a complementary a. The strands are oppositely oriented, so that if one strand reads from left to right, the other reads from right to left. Such base-pairing is the key to the faithful replication of DNA, in which the strands separate and then serve as templates for new complementary strands.

When a gene is expressed, a region along one strand of the DNA is transcribed by the enzyme RNA polymerase. The resulting molecule of RNA is also a chain of nucleotides, and it is complementary to the transcribed strand of DNA. RNA is chemically quite similar to DNA, except that thymine is replaced by the closely related base uracil, so that the RNA alphabet consists of a, c, g and u. Although RNA is single-stranded, base-pairing nonetheless has an important place in its chemistry: Complementary pairing between regions of the strand determines how the RNA folds up to form a distinctive three-dimensional structure.

The RNA transcribed from most genes is not an end product but rather serves as an intermediary, called messenger RNA, which is subsequently translated into protein. Proteins are another class of linear molecules, but they are assembled from a wholly different alphabet, namely 20 amino acids. Each sequence of amino acids folds up to form a specific three-dimensional structure, guided by chemical interactions that are even more complex than those observed in DNA and RNA. The translation from RNA to protein is done by ribosomes, which are large molecular assemblies made up of both protein and RNA; amino acids are carried to the site of translation by transfer RNAs. The ribosomal RNAs and transfer RNAs are examples of RNA molecules that are not themselves translated into proteins. Even so, like proteins, they derive their functionality in large degree from the shape of their folded structure.

In translation, RNA can be interpreted as a language of triplets, in which successive groups of three adjacent bases—called codons—specify the sequence of amino acids in a protein. This is the language that biologists had in mind when they first began to speak of a genetic code and of transcription and translation. Four letters taken three at a time yield 64 possible codons, all of which are used, although there are only 20 amino acids spelled out in the genetic code. It follows that the code must have a good deal of redundancy, where several codons all specify the same amino acid. A few codons serve as marks of punctuation, signaling where translation should start and end.

The RNA polymerase molecule can be viewed as a simple linguistic transducer, which reads bases of DNA and writes complementary bases in the slightly different alphabet of RNA. Indeed, the enzyme is remarkably machine-like in its procession down the DNA template and synthesis of RNA output. Similarly, the ribosome is a transducer from RNA to protein. The ribosome begins by scanning an RNA molecule one base at a time, looking for the codon aug, which is the start signal of the genetic code and which also specifies the amino acid methionine (*met*).

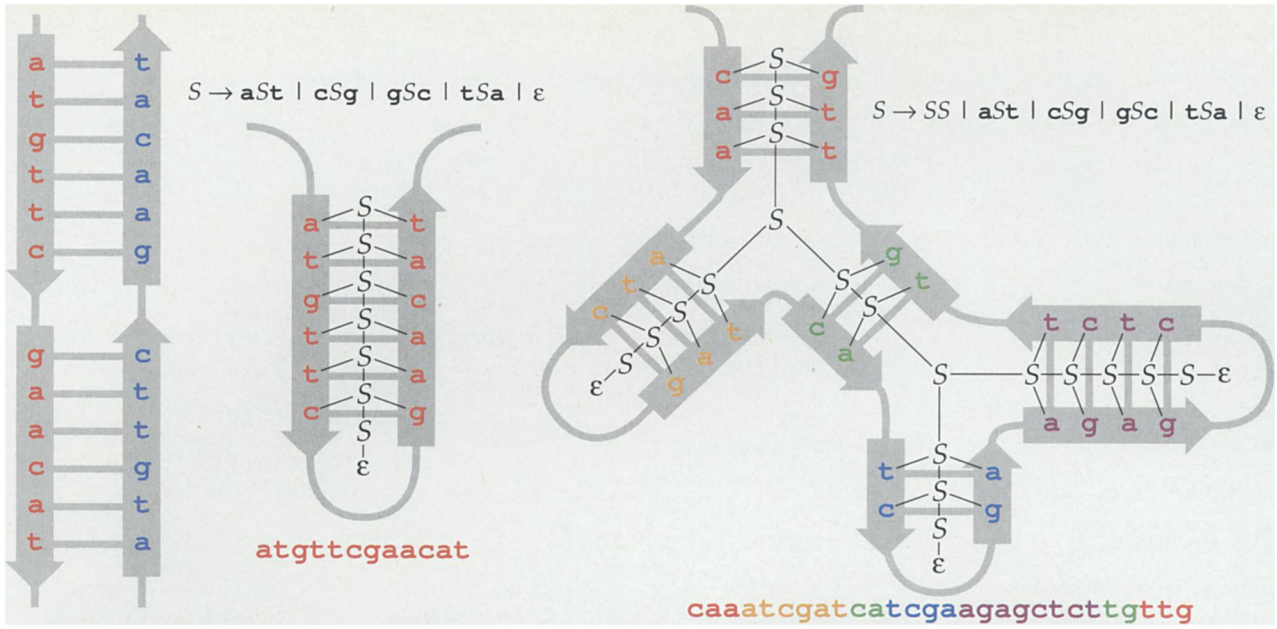

Figure 10. Biological palindromes in the genome give rise to distinctive secondary structures in folded molecules. In double-stranded DNA *(far left)*, each g pairs with a c on the opposite strand, and each t with an a. When a substring of one strand appears on the other strand running in the opposite direction, the resulting pattern is called an inverted repeat. The symmetry of the pattern allows either of the strands to fold and pair with itself *(middle)*; the corresponding single-stranded RNA can adopt the same stem configuration. The language of such biological palindromes is specified by a context-free grammar. (In real nucleic acids there is a loop of unpaired bases at the tip of the stem; the grammar is easily extended to accommodate such features.) By adding a rule that doubles the start symbol, the grammar is able to generate strings that form branched secondary structure *(right)*, as is often found in RNA molecules. The parse tree for any derivation of these grammars reflects the actual physical structure of the folded RNA and is shown here drawn within the structure.

On finding an aug, the transducer outputs a *met* unit, then continues scanning in a mode where it looks at groups of three successive nucleotides. For each triplet, the transducer (with the help of transfer RNA "adaptors") adds a specific amino acid to the growing protein chain; for example, gcg corresponds to alanine (*ala*), and aag to lysine (*lys*). The scanning continues until the transducer comes upon one of the triplets uaa, uag or uga, which do not specify an amino acid but instead terminate the translation process.

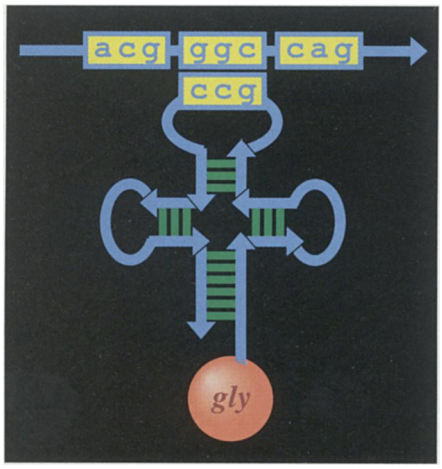

Figure 11. Cloverleaf pattern of transfer RNA is an example of branched secondary structure. The loop of the topmost stem includes a triplet complementary to a codon recognized on the messenger RNA. This codon specifies the amino acid carried at the bottom of the main stem of the transfer RNA.

If these two transducers (as described so far) could be combined and set loose on a real genome, they would eventually translate into protein all the coding regions present in the DNA. Unfortunately, the transducers would also produce an enormous volume of utter nonsense. One problem is that although every protein starts with a methionine unit, not every methionine appears at the start of a protein chain. Furthermore, each strand of DNA has three "reading frames," distinguished by whether the transducer begins reading with the first, the second, or the third nucleotide; since a DNA molecule is double-stranded, there are six reading frames altogether. Each of the reading frames generates a totally different message, and with few exceptions only one such transcript is valid. Thus, the actual transducers must be guided to the correct sites and the correct reading frames by other factors. It is worth remembering in this regard that DNA is not simply an abstract string of symbols but rather a molecular object in a cellular context. For example, it spends much of its existence tightly wound around proteins like thread on a succession of spools, and it interacts with a great many other proteins at specific sites.

The reality is no less complex for RNA. Even when the ribosome transducer happens to find a valid start signal in the correct reading frame, it cannot be taken for granted that the resulting protein is an actual gene product. In higher organisms genes are generally discontinuous, with long stretches of noncoding verbiage that must be spliced out in a step called processing. The meaningful regions that are preserved for translation are called exons; the intervening excised regions are introns. During transcription the entire length of the gene, including both exons and introns, is copied into RNA, but processing must be completed, to remove the introns, before the RNA can be translated into protein.

The removal of introns is largely governed by specific sequences in the RNA transcript. In some genes in some organisms these sequences participate in a characteristic folding of the RNA and are sufficient to remove the intron without outside help. Protein-coding genes rely on a more involved mechanism. The intron signals are found at and near the ends of the introns to be spliced, and they are recognized and bound by a complex of RNA and protein that supervises the precise removal of the intron. The sequence at the upstream end of the intron includes, among other features, the two-base sequence gu. The downstream end of the intron has sev-

eral landmarks, ending in an ag at the splice site itself.

Although a simple transducer could perhaps be designed to model at least the bare facts of intron removal, this would miss the point that splicing involves complex structures and interactions that would be better described by a more hierarchical representation. In fact, the wholesale rearrangement of specific sequences is reminiscent of linguistic transformations that manipulate parse trees. What sort of grammar would be necessary to produce useful parse trees, and to be more selective in recognizing protein-coding genes?

A Grammar of Genes

A grammar describing coding sequences as they appear in the DNA would capture the genetic code in a straightforward manner, recursively building up strings of codons beginning with a *met* codon and ending with one of the stop codons. The rules mapping codons to amino acids would constitute the lowest, lexical, level of the grammar. Introns might be inserted at any point during this accretion of codons, but there is a complication: Processing of the RNA is not constrained to a particular reading frame, so that it is quite possible for an intron splice site to interrupt a codon. One way to accommodate such interpolations is with context-sensitive grammar rules that give rise to movement of nonterminals in the developing string.

The part of the gene grammar dealing with translated sequences would be embedded in a higher-level rule for the entire RNA transcript, including transcribed but untranslated regions at both ends of the RNA; and the transcript rule would in turn be embedded within a rule of still wider scope covering upstream and downstream control regions associated with the gene. This rule structure allows for a natural hierarchical organization of our knowledge about the mechanisms of gene expression, with detail always presented at its appropriate level. What is most challenging at this point is the incorporation of grammar elements for the subtle signals controlling which potential coding regions are expressed as genes, and how they are processed.

Important features that signal the presence of a protein-coding gene, which are collectively called the promoter region, lie upstream of the transcript itself. In this region is a sequence called the TATA box, after its characteristic nucleotide sequence tata. In higher organisms many other sequences may be present, such as a CAAT box further upstream—but with more variability—and GC boxes, which can be found in multiple copies on either strand of the DNA. Promoter sequences are involved in the binding of RNA polymerase at the start of transcription. Their effectiveness is greatly influenced by the presence of other genetic elements called enhancers, whose positions and orientations relative to the gene vary enormously; they may lie thousands of bases away, upstream or downstream.

The variation in the sequences of promoters, enhancers, splice signals and so on has made them hard to identify reliably. Most of them have been discovered either by noting similar sequences at similar positions relative to many genes, or else by direct evidence that they are binding sites of other molecules such as transcription factors, the proteins that mediate gene expression. In either case, the linguistic problem is one of recognition. From one perspective, the challenge is to recognize a "word" similar to a statistical aggregate of examples called a consensus sequence. From the other perspective, recognition models the action of a transcription factor (for example) in finding and binding to the appropriate sequence. Although grammars have their shortcomings in this regard, they also have some great strengths.

A recurring theme in these recognition regions is that more than one sequence comes into play at once. Transcription factors seldom act alone; instead a number of them seem to be required to work cooperatively. Similarly, in the processing of introns several RNA-protein complexes bind different sites. Grammars, by their nature, describe the relationships of words, and of categories of words, and of categories of categories. This last point is important because some transcription factors recognize not a DNA sequence directly but rather other transcription factors already bound to specific sequences. The picture that emerges is of a complex of transcription factors being modeled by a parse tree describing the organization of those factors and of the sites to which they bind on the string of DNA.

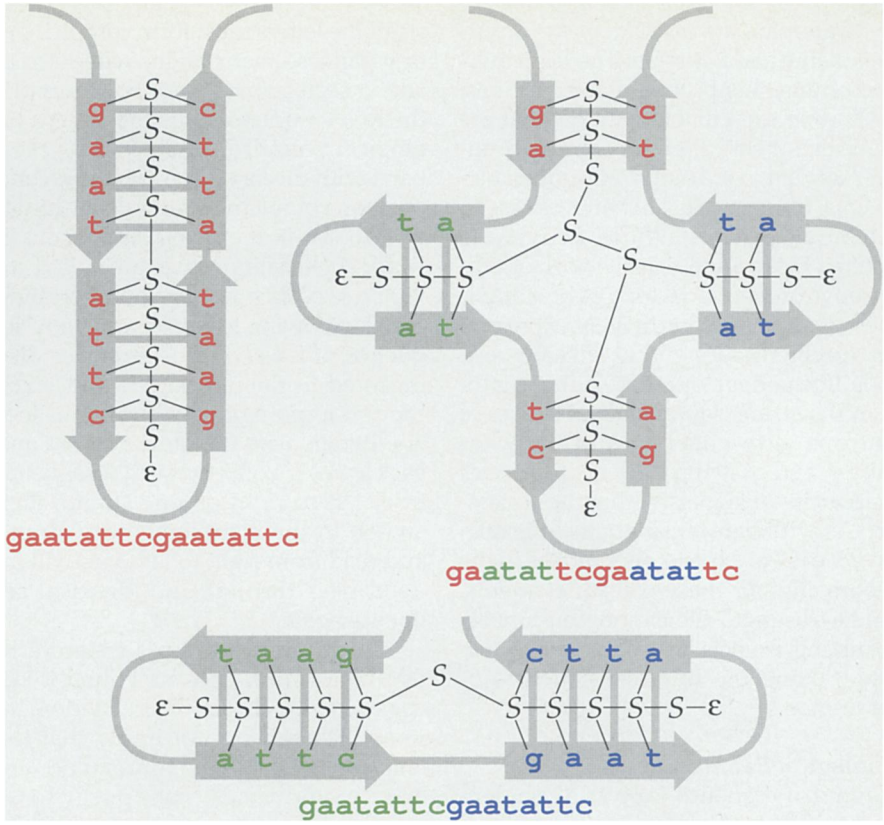

Figure 12. Ambiguity in the grammar for nested palindromes gives rise to strings that have more than one parse tree, or molecules with more than one secondary structure. A double inverted repeat can form a simple hairpin *(upper left)*, an intermediate cruciform structure *(upper right)* or a dumbbell *(bottom)*. Although there is an unambiguous grammar for the language of general secondary structure, the ambiguous grammar may be preferable since the alternative parse trees reflect alternative secondary structures. Moreover, any grammar that specifically generates only adjacent pairs of inverted repeats must be ambiguous.

Grammars are also adept at capturing theme and variation. There are myriad arrangements of regulatory regions, including identifiable patterns that are specific for sets of genes expressed under similar conditions. Response elements, for example, are patterns associated with genes whose activity is altered by hormones, heat shock or other environmental factors. Whole families of genes are coordinately expressed in specific tissues and at specific periods in the development of the organism. There are classes and subclasses of introns with varying splice mechanisms, and growing evidence of species specificity of signals within these classes. As with natural language, it is relatively easy to express alternatives in the "phrasing" of these control elements, and to distinguish the more constant organizing aspects from the more variable, using the implicit structure of grammars.

Biological Palindromes

Grammars can also express dependencies with great facility. Such dependencies are likely to be inherent in regulatory regions, as we have seen, but dependencies may also be present even in the actual lexical sequences of DNA and RNA.

Many protein-binding sites on DNA are palindromes, although they differ somewhat from natural-language palindromes. The requirement for a biological palindrome is not that the letters be identical when they are matched one by one from both ends of the string; instead the letters must form complementary pairs, so that a aligns with t (or u) and c corresponds to g. For example, the RNA sequence aguucgaacu is a biological palindrome, since the first a pairs with the last u, the g in the second position complements the c that comes next to last in the string, and so on. In DNA such palindromes are called inverted repeats and have an interesting corollary, owing to the fact that any sequence of bases in one strand is also mirrored in the opposite strand. Even though a biological palindrome does not literally read the same forward and backward, if you read such a sequence from left to right on one strand, then switch to the complementary strand and read from right to left, you will indeed pass through the identical sequence again.

Palindromes are quite frequent in DNA, and short ones are bound to occur simply by chance. These random inverted repeats do not imply that the language of DNA is nonregular, and even palindromes that appear to be purposeful (such as those associated with protein binding) may yet be regular, technically, if there is a definite limit on their size. It is only when a language requires dependencies over an arbitrary extent that it must be deemed greater than regular.

The folding and base-pairing of an RNA strand to form what is called secondary structure does establish dependencies between the paired bases, and there is no theoretical limit on the distance over which those dependencies might extend. An RNA palindrome can be folded in the middle to create a base-paired "hairpin" structure, which is thermodynamically more stable than the same RNA without base pairing. Moreover, one cannot identify a longest hairpin, such that it would not be possible to add just one more base pair. To the extent that such folding is a requirement of the language of nucleic acids, they are beyond regular languages by virtue of their very chemistry.

A grammar for RNA palindromes is only a little more complicated than a grammar of binary-numeral palindromes. The underlying principle is the same, although the alphabet is larger and the nature of the pairing is different. Here is one form of the grammar:

$$S \rightarrow aSu \mid uSa \mid cSg \mid gSc \mid \varepsilon$$

Note that the nonterminal S is embedded within a string in the productions on the right-hand side of the rule, which is the trait that distinguishes a context-free grammar from a regular grammar. Trying a few random derivations gives a sense of how the grammar can generate all RNA palindromes, and only palindromes, as in this example:

$$S \Rightarrow aSu \Rightarrow auSau \Rightarrow augScau \Rightarrow augcau$$

Folding the final structure in half and pairing the bases that are thereby brought together yields a perfect match. What's more, the parse tree produced has the pleasing property that it can be drawn to reflect the base pairing seen in the actual secondary structure.

The perfection of the palindromes

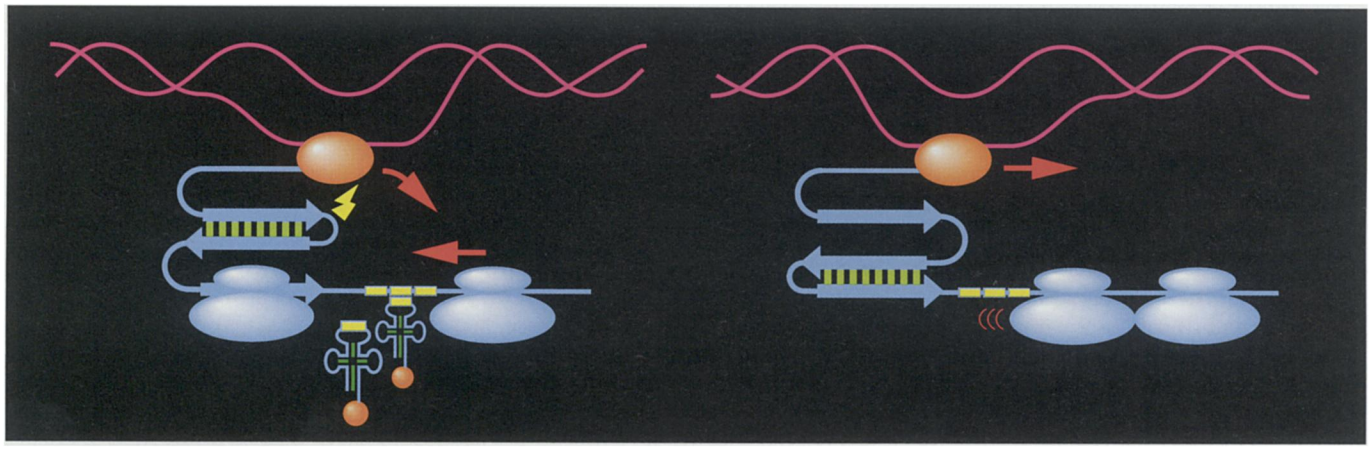

Figure 13. Attenuators are regulatory mechanisms that are thought to depend on alternative secondary structure for their operation. They are found upstream of certain bacterial genes coding for proteins that help to manufacture amino acids. When RNA polymerase begins to transcribe this upstream region, ribosomes immediately attach to the RNA and begin to translate the sequence. If the amino acid to be synthesized is present in abundance (and thus the corresponding transfer RNA is abundant as well), the first ribosome reads through a group of codons for that amino acid and into a region capable of forming either of two alternative hairpins *(left)*. The ribosome obstructs the first part of this region and thus favors the formation of the second stem. Formation of the second hairpin sends a signal that causes the RNA polymerase to cease transcribing and to fall off the DNA. This shuts down gene expression and the synthesis of the amino acid (which was already abundant). If the amino acid is scarce, the ribosomes cannot read through the codons for that amino acid *(right)*; they stall upstream and allow the first alternative hairpin to form instead of the second. This event in turn allows the RNA polymerase to proceed and manufacture the protein that will alleviate the shortage of the amino acid.

generated by this grammar is actually a weakness from the point of view of biological realism. Real RNA palindromes are often imperfect, but an occasional mismatched base pair does not destroy the secondary structure. (Some alternative base pairings, such as g:u pairs, are more tolerated than others.) It is common for the stem of a hairpin to have "bulges" of unpaired bases. Moreover, the RNA is not flexible enough for the tip of the hairpin to make a sudden 180-degree turn; generally there is a loop of at least three or four unpaired bases, and sometimes the loop is much longer. Similarly, in DNA significant inverted repeats may be separated by very large distances. Incorporating such features into a grammar complicates the form of the rules, but it can be done, for example by substituting for the ε in the palindrome grammar a nonterminal representing the loop, and adding a simple, regular rule specifying that loop. It is perhaps more interesting, though, to study further the linguistic nature of idealized secondary structure with complete base pairing.

All of the palindrome grammars presented so far are limited to generating strings containing a single palindrome whose center of symmetry is the middle of the string, as in the classic "Madam, I'm Adam," where the "I" marks the center of the (odd-length) palindrome. But strings can have multiple palindromes, most simply when two or more palindromes are found side by side in the same string, as in "Madam Eve." Nucleic acids are rich in such compound palindromes. Indeed, DNA and RNA abound not only in sequential palindromes but also in recursive ones, where one palindrome is embedded in another. For example, "Madam Eve, I'm Adam" has the short palindrome "Eve" inserted off-center in the longer one. The secondary structure that corresponds to a recursive palindrome is a stem with another stem budding from its side. In principle there can be any degree of nesting of stems upon stems upon stems.

Modifying the palindrome grammar to allow for recursive secondary structure turns out to be simpler than one might guess. All that is required is to add to the existing productions a new one, stating that $S \rightarrow SS$. Duplicating the start symbol plants the seed of a new palindrome anywhere within an existing palindrome. It can be shown that this is a completely general grammar describing structures of branched stems, known as orthodox secondary structure. The most famous example of orthodox secondary structure is the cloverleaf structure of transfer RNA, but there are many other instances of extensively nested stem-and-loop structures, particularly in ribosomal RNA.

The grammar of recursive palindromes remains context-free, but orthodox secondary structure alters the nature of the language in another way: Ambiguity enters the scene. A grammar

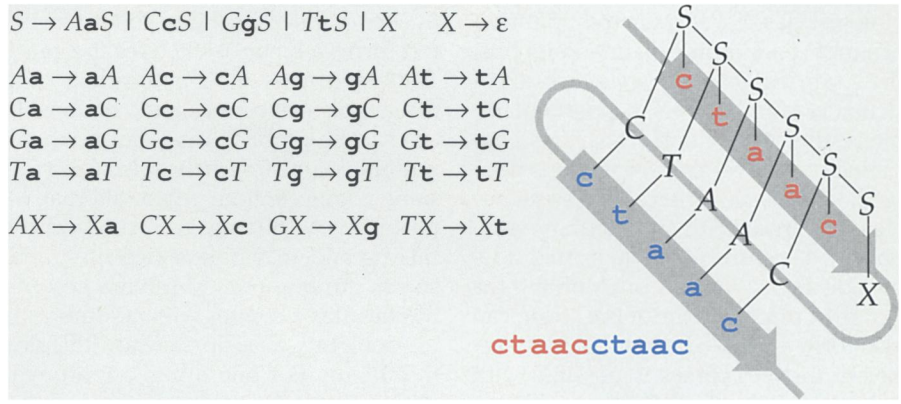

Figure 14. Direct repeats, or simple repetitions of a sequence, are a common feature of DNA. The language consisting of all such sequences is known to be greater than context-free. Here direct repeats are generated by a grammar that produces a nonterminal and a terminal for each base in the repeated sequence, then "slides" the nonterminals to the right where they are converted to terminals in the correct order by interaction with the nonterminal *X*. In terms of the parse tree, the nonterminals are skipped over and then gathered up at the position of the *X* in the tree. This rearrangement creates crossing dependencies.

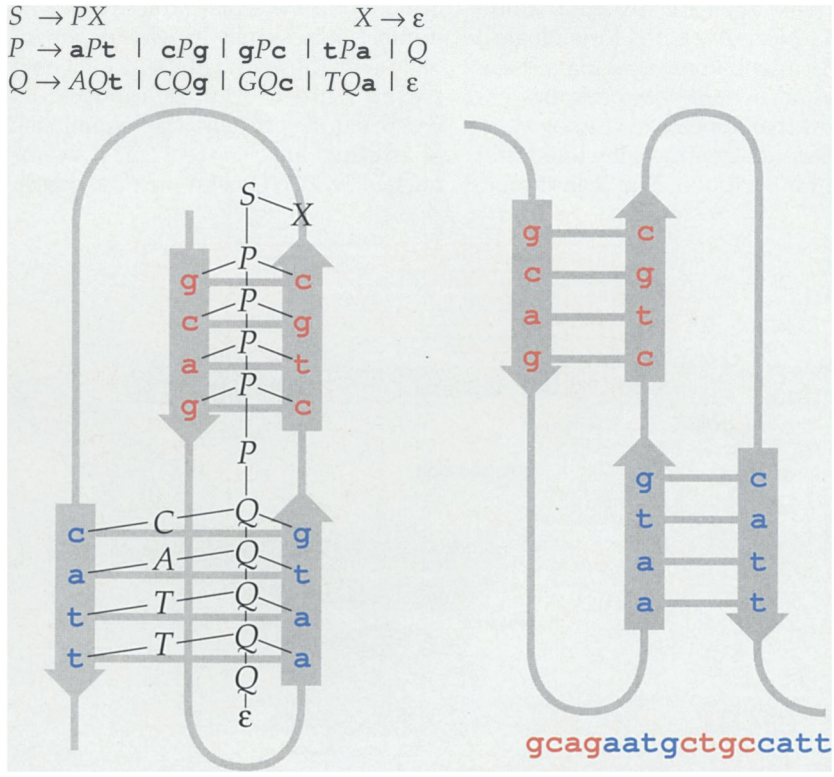

Figure 15. Pseudoknots are elements of secondary structure that may produce crossing dependencies and require non-context-free expression. A pseudoknot in RNA, shown in the diagram at right, results when one side of a stem resides within the loop of another stem. Only the context-free part of the pseudoknot grammar is given; the complete grammar also includes all the context-sensitive rules shown above in the grammar for direct repeats. The pseudoknot grammar generates an idealized pseudoknot language, without any unpaired bases. Like the direct-repeat grammar, it "skips over" the nonterminals produced by the *Q* rules and gathers them up at the position of the *X* in the parse tree. Tracing the terminal string around the parse tree in this manner produces the topologically equivalent secondary structure shown at the left, and preserves the correct base-pairing dependencies.

is ambiguous if it can generate the same string with two different parse trees. Syntactic ambiguity in natural language can be seen in the sentence "The linguist sees the biologist with the telescope," where in one parse the linguist has the telescope, but in the other the biologist has it. Whereas the grammar of simple palindromes is unambiguous, the grammar of recursive secondary structure is ambiguous in a particularly interesting way. Consider strings consisting of an inverted repeat that is doubled, such as gatcgatc. This sequence clearly parses with the recursive-secondary-structure grammar as two side-by-side stems, formed by doubling the *S* at the outset. But the entire string can also form a simple hairpin. In fact a series of distinct parses is possible with such strings, in which *S*'s are doubled at a variety of points in the derivation. In each case, the parse tree again mimics the actual form of a potential folded structure—the ambiguity of the grammar is indeed modeling a known phenomenon in RNA, that of alternative secondary structure.

In at least one case structural ambiguity in RNA is exploited for biological effect. An attenuator is a regulatory element found in some bacterial messenger RNAs that appears to employ alternative secondary structures to control its own transcription. The palindromic elements of an attenuator can form a stem and loop in either of two mutually exclusive conformations; one base-pairing allows transcription to continue, whereas the other causes it to terminate prematurely. The tendency to one or the other secondary structure, under the influence of the biochemical context, in effect forms a binary switch for the gene. There are many other indications that the language of genes should be considered to be ambiguous—cases where multiple start sites can be selected in the same coding region, where alternative patterns of splicing are achieved by mixing and matching splice sites, and so on. Ambiguous grammars accommodate these themes with aplomb.

Ambiguity, alas, does create difficulty in parsing. For one thing, a grammar with more than one nonterminal on the right-hand side of a production (such as the duplicated start symbol *SS*) is called nonlinear, and it is disqualified from certain speedups allowed in parsing linear context-free grammars. Even worse, if we are interested in alternative secondary structures, we should be interested in all possible parse trees. The number of trees may increase exponentially with the length of the string being parsed. Thus even if each individual parse is quite efficient, the overall task of finding all the trees may be intractable with a general-purpose parser.

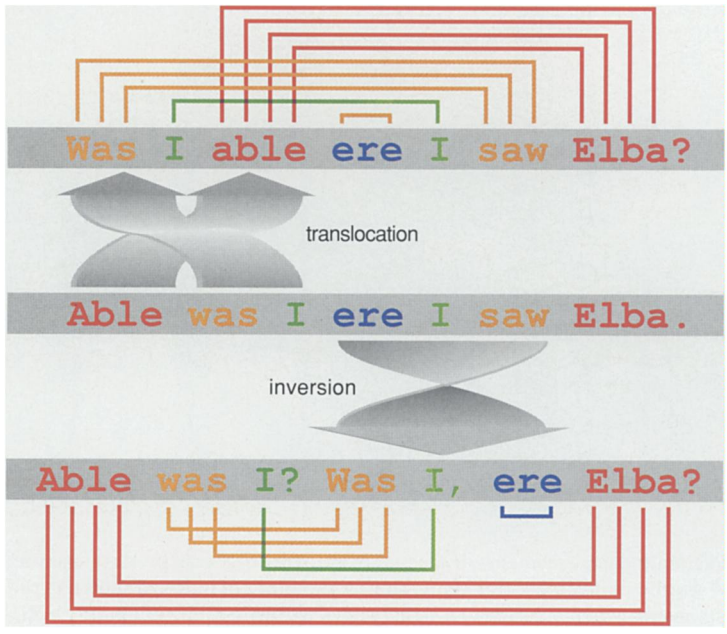

Figure 16. Evolutionary "operations," such as the translocation or inversion of genetic sequences, may promote a language to a higher level in the Chomsky hierarchy. The strictly nested dependencies of the palindrome at the center become crossed dependencies when a translocation occurs, as in the word re-ordering at the top. This creates a pseudoknot-like structure. An inversion within the palindrome, as at the bottom, may create a direct repeat, also with crossing dependencies.

Climbing Chomsky's Ladder

All of the palindromic structures, no matter how elaborate, remain within the family of context-free languages. But there are features of nucleic acids that do lie beyond the descriptive power of context-free grammars. The most obvious example is the presence of direct repeats in DNA. Direct repeats are a fairly common motif in the genomes of most organisms; for example, they occur in some enhancers, in segments of DNA that enter and leave the genome (such as viruses) and in the "amplification" of certain heavily used genes. Direct repeats may arise when a segment of DNA is duplicated, and they essentially comprise a copy language. Hence they are beyond context-free, provided they are essential to the language and are not simply present by chance.

A more interesting example of a non-context-free structure is found by referring again to the mechanism of RNA folding. RNA structures called pseudoknots can be understood as palindromes that are interleaved rather than nested. For example, where aaccgguu can be seen as a nesting of the two biological palindromes aauu and ccgg, the sequence aaccuugg is a pseudoknot, where the complementary pairings must cross over each other in order to form a base-paired secondary structure. An English example would be "DNA's loops and spools," where "DNA" and "and" form one palindrome and the remainder of the string another. In actual pseudoknots, which are observed (for example) in certain RNA viruses and in a class of introns, one side of the stem of a stem-and-loop resides within the loop of another stem-and-loop. Features of this kind are referred to as nonorthodox secondary structure. As with direct repeats, pseudoknots entail crossing dependencies that exceed the capabilities of any pushdown automaton. A pseudoknot parser might push "DNA's loops" on a stack, but then it could not pop "DNA" from the bottom of the stack (in order to generate "and") without discarding the intervening letters that would be needed later for "spools." Both direct repeats and pseudoknots can be described by context-sensitive grammars and the corresponding automata.

Specific secondary structures are characteristic of ribosomal RNA and transfer RNA and other forms of RNA that are not translated into protein. In protein-coding genes the evidence for such conserved syntactic structures is

less obvious. Secondary structure in transcripts may be important in splice-site selection in cases of alternative splicing, and an excess of secondary structure at the beginning of RNA transcripts, where ribosomes first attach, may hinder translation in certain instances. In such cases, the absence of stems could be as important as their presence, for example if they interfere with some mechanical process. This is interesting from a mathematical standpoint because context-free languages are not closed under complementation—that is, the language of all strings that are *not* members of a given context-free language need not be context-free. Whether the resulting language of translatable transcripts is in fact greater than context-free on this account depends on the specific secondary structures that were excluded.

Closure arguments suggest a mechanism by which the language of genetic sequences may have ascended the Chomsky hierarchy, from the first random (and thus regular) polymers that may have condensed in the primordial soup. For the everyday operations that are performed on DNA—replication, scission, ligation—all the levels of the Chomsky hierarchy are closed. For example, creating the complementary strand of a string of DNA cannot make a context-free language greater than context free (although it *can* make an unambiguous language ambiguous). However, context-free languages are not closed under *evolutionary* operations—those that involve block movements of segments of a genome. This is easily seen in the case of duplication, which creates direct repeats. It is less obvious why a context-free language of DNA should not be closed under inversion (where a segment of double-stranded DNA is flipped to an opposite orientation) or transposition (where two segments exchange positions). But consider what effect these operations can have on a palindrome: By reversing half of a palindrome, inversion creates a direct repeat, whereas transposition can produce a pseudoknot structure.

Another source of linguistic complexity in the living cell is superposition. A gene grammar must reflect a number of processes mediated by separate machinery at different times in different parts of the cell. The language of genes really represents the intersection of separate languages for transcription, processing, translation and even for the encoded protein sequence, which must specify the folded structure of the protein with its rich array of nested and crossing dependencies. Context-free languages, it will be recalled, are not closed under intersection.

Practical Parsing

The recognition of genes and other high-level features of DNA sequences is challenging from both a theoretical and a practical standpoint. Even if the language of DNA is greater than context-free, a practical parser can include special-purpose features that anticipate known problematic aspects of the language, such as direct and inverted repeats. This strategy can ameliorate the inefficiency of general-purpose parsing, at least for appropriately constructed grammars. Balancing this gain is the need to examine a potentially exponential number of parses for some grammars, and the fact that many recognition sites are underconstrained and vary in "strength."

In my own work, I have developed a parser in the logic programming language Prolog for a new grammar formalism that is "slightly" greater than context-free and that handles repeats in any form. This allows me, for example, to write a grammar for transfer RNAs that performs on a par with specialized algorithms designed for the purpose, and that offers much greater flexibility to modify and add rules—another strength of grammars. I have also added mechanisms that allow for imperfect matching and for keeping track of the "goodness of fit" of the parse. For protein-coding genes, where the combinatorics of splicing leads to a prohibitively large number of parses, I am using the grammar as a framework to apply statistical heuristics that others have used to detect likely coding regions, in this way limiting the parser to only the most probable combinations.

Such enhancements necessarily depart from the purity of formal grammars and from the realm of syntax, delving instead into the semantics, or meaning, of language. English syntax allows sentences that are nonsensical, even though grammatical, just as the rudimentary gene grammar described here allows nonsense proteins. Typical natural-language understanding programs hand off the results of parsing to components that attempt to evaluate the meaning of an utterance, perhaps in a much larger context, and thus help to resolve any syntactic ambiguity. In a similar way, gene grammars offer a means to encapsulate that which is inherent in the biochemistry of nucleic acids and the machinery that impinges on them, and then to incorporate more concrete data, heuristic methods or biochemical context in a principled way. In the process, the grammars developed serve to help codify our current knowledge of the higher-order structure of genetic information, and classify its complexity in terms relevant to its computational analysis.

Acknowledgments

For useful discussions and contributions to this work, the author thanks Brian Hayes, Erik Cheever, G. Christian Overton, James Tisdall, Sandip Biswas and Shan Dong. The author is supported in part by the Department of Energy under Genome Grant 92ER61371.

Bibliography

Brendel, V., and H. G. Busse. 1984. Genome structure described by formal languages. *Nucleic Acids Research* 12:2561–2568.

Brendel, V., J. S. Beckmann and E. N. Trifinov. 1986. Linguistics of nucleotide sequences: Morphology and comparison of vocabularies. *Journal of Biomolecular and Structural Dynamics* 4:11–21.

Collado-Vides, J. 1989. A transformational-grammar approach to the study of the regulation of gene expression. *Journal of Theoretical Biology* 136:403–425.

Head, T. 1987. Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology* 49(6):737–759.

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

Lewin, B. 1990. *Genes IV*. Cambridge, Mass.: Cell Press.

Searls, D. B. 1988. Representing genetic information with formal grammars. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 386–391. San Mateo, Calif.: Morgan Kaufmann.

Searls, D. B. 1989. Investigating the linguistics of DNA with definite clause grammars. In *Logic Programming: Proceedings of the North American Conference on Logic Programming*, ed. E. Lusk and R. Overbeek, pp. 189–208. Cambridge, Mass.: MIT Press.

Searls, D. B., and M. O. Noordewier. 1991. Pattern-matching search of DNA sequences using logic grammars. In *Proceedings of the Seventh Conference on Artificial Intelligence Applications*, pp. 3–9. Los Alamitos, Calif.: IEEE Computer Society Press.

Searls, D. B. 1992. The computational linguistics of biological sequences. In *Artificial Intelligence and Molecular Biology*, ed. L. Hunter, pp. 47–120. Cambridge, Mass.: AAAI Press.

Searls, D. B., and S. Dong. In press. A syntactic pattern-recognition system for DNA sequences. In *Proceedings of the Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, ed. H. A. Lim, J. Fickett, C. R. Cantor and R. J. Robbins. Singapore: World Scientific.