Abstract

Biologists have long made use of linguistic metaphors in describing and naming cellular processes involving nucleic acid and protein sequences. Indeed, it is very natural to view the genetic 'text' and its sequential transliterations in these terms. However, a metaphor is not a tool, and it is necessary to ask whether the techniques used in analyzing other kinds of languages, such as human and computer languages, can in fact be of any use in tackling problems in molecular biology. This paper reviews the work of the author and others in applying the methods of computational linguistics to biological sequences.

Introduction

Only in recent years has the long-standing metaphor of DNA as language been rigorously examined, from both theoretical and practical perspectives. This metaphor arose early in molecular biology, when nucleic acids were recognized as strings of nucleotide bases comprising the famous four-letter alphabet. The complementarity of this alphabet—with the letter G always pairing with C across the double helix, and T with A—was seen to permit the faithful replication of DNA molecules. The metaphor was strengthened when the relationship of nucleic acids to proteins was elucidated; the so-called central dogma recognized the fundamental two-step process that first transcribes a subsequence of DNA into RNA, and then translates successive triplets of RNA bases to amino acids according to the mapping called the genetic code. Later, it was discovered that the RNAs that are translated, called messenger or mRNAs, are further processed in higher organisms so as to splice out intervening untranslated regions, called introns, leaving only the exons of the ultimate transcript. These biological transformations can be seen as analogous at a number of levels to mechanisms of processing other kinds of languages, such as natural languages and computer languages, particularly in the approaches pioneered by Noam Chomsky in his revolutionary work in the field of linguistics (Chomsky, 1957).

A comprehensive review of this branch of linguistics is well beyond the scope of this review, but a few examples may serve to highlight some of the relevant issues. Consider the following sentence:

The biologist that the linguist noticed eventually waved.

The lines above the sentence serve to connect each noun with its corresponding verb, i.e. it was the linguist who noticed and the biologist who waved; note that the relative clause thus separates remote features of the sentence that are meaningfully related to each other. This sort of 'action at a distance' is very characteristic of natural language and provides much of the motivation for attempts to account for such structure using rule-based systems called grammars. It will be seen that grammars are capable of neatly generating just such hierarchical descriptions of the components of sentences. They also capture another important feature of natural languages, syntactic ambiguity, by which alternative structural interpretations or parses are possible. In this example, the dashed lines below the sentence highlight an ambiguity surrounding the referent of the adverb 'eventually': was the biologist slow to wave to the linguist, or was the linguist slow to notice the biologist?

The dependencies between nouns and verbs in the sentence above are nested, insofar as any number of relative clauses could theoretically be inserted within each other to create tiers of such dependencies. Somewhat less common in English are crossing dependencies; an artificial example would be the following:

The biologist and linguist noticed and waved, respectively.

As will be seen in the next section, whether a language entails strictly nested or crossing dependencies has significant consequences for the types of grammars required, with greater complexity attaching to crossing dependencies. It is noteworthy that very subtle lexical changes (i.e. at the level of the words of a sentence) can have drastic effects on the parse. In the following pair of sentences, changing the preposition 'by' to the conjunction 'as' completely rearranges the dependencies in the underlying parses:

The biologist noticed by the linguist waved.
The biologist noticed as the linguist waved.
To appreciate the relevance of these linguistic issues to biological sequences, consider the various kinds of structure found in the latter. By analogy with sentences, it might be claimed that genes have a hierarchical syntactic structure, and indeed it is natural to draw tree-structured descriptions of gene structure (Searls, 1993). Just as sentences exhibit distant dependencies, e.g. between related nouns and verbs, so do genes exhibit such dependencies, e.g. between splice donors and acceptors. In this case, syntactic ambiguity would be a reflection of the phenomenon of alternative splicing; it is well known that subtle changes in 'lexical' signals in mRNA can result in patterns remarkably like the example sentences immediately above. Similar ambiguities can be found in the apparatus regulating gene expression.

Another analogy to linguistics arises from the fact that biological strings fold up in three-dimensional space, in such a way that distant parts of those strings interact with and thus create dependencies with each other. In RNA, the most obvious manifestation of such dependencies is base-pairing interaction. As will be seen in the next section, such dependencies are very naturally expressed via grammars. Again, the notion of syntactic ambiguity has a biological manifestation, in this case the phenomenon of alternative secondary structure (Searls, 1993). Both nested and crossing dependencies are observed, for example in antiparallel and parallel strands in proteins (Mamitsuka and Abe, 1994).

The author has discussed a number of other correspondences between linguistic notions and biological phenomena (Searls, 1992, 1993), and has also built tools for pattern-matching search based on linguistic parsers (Searls and Noordewier, 1991; Searls and Dong, 1993). The latter have extended to the problem of detection of genes in genomic DNA sequences (Dong and Searls, 1994). These three topics will be discussed in the following sections of this review. Readers with interest and background in the field of formal language theory, the bare rudiments of which are introduced in the next section, may wish to refer to developments in that arena that have been inspired by the biological domain (Searls, 1989, 1995a,b, 1996); this work is also summarized in a recent review (Searls, 1997).

Formal language theory and biological sequences

The processes of transcription and translation from strings of one kind to strings of a different kind by processive cellular machinery suggested to some the behavior of certain kinds of finite state automata (FSAs). FSAs are simple models of computation, in fact lying at the foundation of computer science, that comprise directed graphs with labeled transitions among states. By traversing such a graph from state to state and emitting the lexical/alphabetic labels upon each transition, a variety of strings can be generated constituting a language. Brendel and co-workers exhibited FSAs that, on paper, modeled the processive processes implied by the central dogma (Brendel and Busse, 1984), and in fact such automata are implicit in numerous software packages that perform sequence analysis. For that matter, such packages often provide capabilities for pattern-matching search for short substrings of interest, through the use of regular expressions (such as the UNIX grep utility); in formal language theory, the most basic form of regular expressions corresponds to FSAs in terms of expressive power. Note that FSAs have a dual nature: they can be seen either as generators of languages, or as recognizers. A variation on this architecture, called a finite state transducer, can accomplish both at the same time; by labeling each transition with both an input and an output, a true transformation of one string to another can be accomplished, even more effectively modeling the process of gene expression.

Given the felicity of this correspondence between FSAs and biology, the question thus naturally arose as to exactly where DNA resides on the Chomsky hierarchy of formal languages. This hierarchy classifies the linguistic complexity of languages (viewed purely as sets of strings) and relates them to species of automata required to recognize or generate them. Similarly, each level of the hierarchy corresponds to a particular type of grammar, or rule-based system for formally specifying languages. It happens that regular languages, those that can be specified by FSAs and/or by regular expressions (a kind of grammar), occupy the lowest level of the Chomsky hierarchy. However, there are certain languages, such as the set of palindromes (strings that read the same forward and backward), that cannot be expressed by any FSA or pure regular expression. Fundamentally, this is because FSAs and regular expressions have no notion of memory that would permit them to describe arbitrary numbers of dependencies, other than strictly local ones.

Languages such as palindromes fall on the next level of the Chomsky hierarchy, that of context-free languages, which require a pushdown (stack) automaton and a more powerful form of grammar consisting of rewrite rules. In this system, the alphabet is augmented with a set of temporary, placeholder symbols or non-terminals, and a string is derived by starting from some such symbol and applying the rewrite rules to non-terminals in the developing string until they are all eventually replaced by a terminal string of alphabetic elements only. Thus, for example, the grammar consisting of the rules

\[ S \rightarrow gSc \quad S \rightarrow cSg \quad S \rightarrow aSt \quad S \rightarrow \epsilon \]

specifies a set of DNA molecules. The \( S \) is a non-terminal, while the lower-case letters are terminal bases; the \( \epsilon \) stands for the empty string, and effectively serves to erase an \( S \). This grammar specifies an infinite number of strings via derivations like the following, in which the \( S \) is repeatedly rewritten...
using the above rules:

$$S \Rightarrow gSc \Rightarrow gcSc \Rightarrow gcSatgc \Rightarrow gcataatgc$$

The reader may confirm, by drawing lines connecting those nucleotides that were derived by the same rule invocation, that this grammar creates strictly nested dependencies, and in fact nested dependencies are the only sort possible from a context-free grammar. As noted, the resulting language lies outside the regular languages, though any regular language can be expressed by a grammar like the one above. Nevertheless, there are yet other languages that elude even the context-free formalism, in essence because they entail arbitrarily many crossing dependencies. Many of these can be captured with a context-sensitive grammar: one that can have more than one symbol on the left-hand side of a rule, as long as the number of symbols does not exceed that on the right-hand side. Relax this latter restriction, and the resulting unrestricted grammars are able to specify any language that can be recognized by a Turing machine—the automaton corresponding to this class of languages. Thus, ascending the Chomsky hierarchy appears to require more and more computational power to accomplish general-purpose recognition or parsing of strings, i.e. to determine their membership in a language specified by some grammar.

Much of the author’s work has been concerned with the formal characterization of the language of nucleic acids, in terms of its position in the Chomsky hierarchy and related mathematical questions. For example, the grammar given above specifies, in an idealized way, an important class of biological sequences, those exhibiting dyad symmetry. The strings of this language constitute, in fact, a variety of biological palindrome in which a string reads the same on one strand of DNA as it does on the opposite strand reading backward (the so-called reverse complement). The resulting inverted repeats may also allow a single strand to fold up and base pair with itself instead of with its opposite strand, in a structure called a hairpin or, when there are a number of unpaired bases at the turn, a stem-and-loop. Such secondary structure is crucially important in, for example, the function of structural RNAs in the cell. To the extent that the capacity for secondary structure may be said to be a necessary feature of the language of nucleic acids, we may infer that they lie above the regular languages on the Chomsky hierarchy, since at least a context-free grammar is required to specify such structure in the general case. In fact, the most general grammar of orthodox secondary structure can be shown to consist of the grammar above, augmented with one additional rule: $S \rightarrow SS$. This rule, which simply doubles the start symbol, is sufficient to permit arbitrarily branching secondary structures (Searls, 1989, 1993). It also has the effect of introducing the potential for ambiguity, in that there are sequences that can be parsed by more than one path [the reader may wish to verify this by trying parses with this grammar on sequences that are double inverted repeats, such as gatcgatc; these can theoretically form hairpins, dumbbells, or even cruciform structures, each corresponding to a different parse (Searls, 1992)]; happily, these correspond to alternative secondary structures, providing another useful analogy between linguistic theory and biological reality (Searls, 1992, 1993). Stochastic forms of such grammars, i.e. where probabilities are attached to each rule, have proven very successful in machine-learning approaches to characterizing recurring secondary structures, as in the work of Haussler’s group (Grate et al., 1994; Sakakibara et al., 1994).

Still other biological phenomena indicate that the language of nucleic acids may be beyond context free as well. Pseudoknots (see below) are forms of secondary structure that require context-sensitive expression in the general case, as do repeated sequences that are common in DNA and are arguably necessary features of the language. Closure properties and decidability results suggest that evolution may be a powerful force toward increasing linguistic complexity, and that DNA may be inherently ambiguous, non-linear and non-deterministic (all formally defined properties from language theory). These results are described in Searls (1992), and presented in greater mathematical detail in Searls (1993, 1997); in the remainder of this review, we will address the more pragmatic problem of recognition or parsing of such linguistic features in DNA.

In speaking of higher-order pattern recognition in biological sequences, this linguistic point of view offers one way to categorize the problem space and suggests established tools for exploring it. For example, the existence of features in the domain that are at least context free implies that simple regular expression search will not suffice. Even in advance of any formal linguistic analysis, however, this problem was recognized, and a number of software packages have offered enhanced regular expressions with ‘escapes’ to specify, for example, inverted and direct repeats (Staden, 1990). Some early programs for recognizing patterns of motifs in proteins were based on extended regular expressions (Lathrop et al., 1987). Whether or not these offer sufficient expressive power for a greater range of biological phenomena, the use of more sophisticated grammars and parsers can be seen to have other advantages. Perhaps most notable among these is the ability to create modular, hierarchical rule sets, with detail always presented at the appropriate level (and, of course, a well-studied formal foundation). Parsers also typically return tree-structured descriptions of the history of a derivation, called parse trees, which are not only clear depictions of the presumed structure under study, but are also appropriate data structures for further computational analysis. (One worker, in fact, has studied genetic structures from the point of view of transformational grammar, which entails operations on a presumed canonical parse tree in order to create variations in surface structure typical of natural language (Collado-Vides,
correspond to a base substitution. However, mutations involving mismatches at the level of individual bases (so that the cost is, in effect, the Hamming distance). Biologically, this would correspond to a base substitution. However, mutations involving insertions and deletions are also common; the so-called edit distance between words (or, more often, entire strings) accounts for the total number of such operations needed to transform one string into another. Algorithms for efficiently finding such a minimum edit distance alignment between two strings (allowing gaps) have historically played an important role in computational biology (Waterman, 1988). Again at the level of words, a more sophisticated variation on counting mismatches has been to compile frequency tables for the number of times each base occurs at each position in a number of exemplars; this frequency table is converted by a variety of techniques to a weight matrix which is used to assess the cost of a match over the whole word (Staden, 1984). Yet more sophisticated methods have been applied involving hidden Markov models, connectionist techniques, and the like, demonstrating that ‘higher-order’ structural analysis is important even at the level of word recognition in biology (reviewed by Gelfand, 1995).

Thus, what we have termed the linguistic view of biological sequences involves challenges for both word recognition and syntactic analysis. We will describe practical approaches to both these problems in succeeding sections.

A pattern-matching parser

Logic grammars are a well-studied set of grammar formalisms that are closely related to the logic programming language Prolog. Most Prolog language implementations, in fact, offer the capability to compile definite clause grammars (DCGs) directly into executable code that constitutes a recursive-descent parser for that grammar (Pereira and Warren, 1980). The built-in theorem prover of Prolog in effect becomes the parser, and significant advantage can be taken of the list-manipulation and unification features that are a great strength of logic programming.

In a DCG, a rule such as $S \rightarrow gSc$ would be written $s = [g], s, [c]$. Non-terminals are represented as Prolog atoms and terminals are given inside square-bracketed lists. This would be compiled to the Prolog rule:

$$s \langle S0, S \rangle : S0 = [g|S1], s(S1, S2), S2 = [c|S].$$

Notice that a pair of variable parameters have been added to the non-terminals; these difference lists represent the input string and the remainder of the input string, respectively, after that non-terminal is successfully parsed—in effect, the span of the non-terminal on the input. In other words, the difference lists manage the input string, passing it from element to element, and hiding this ‘implementation detail’ from the user at the level of the DCG itself. Terminal lists in the DCG actually consume elements from the input list, in this case by unifying it (using the = operator) with a list having the terminal(s) as its head and the remainder list as its tail. It can be seen that the difference lists are arranged so that the span of the left-hand side non-terminal is that of the entire right-hand side of the rule. Thus, the rule $S \rightarrow e$, written in DCG form as $s \rightarrow [\ ]$, could be translated as simply $s \langle S, S \rangle$. For the overall grammar, actual top-level calls to $s$ would succeed in forms such as $s \{ [g, g, g, c, c, c, c], [\ ] \}$, signifying that the non-terminal $s$ indeed can span the entire input list.

DCGs also allow the embedding of arbitrary Prolog code in rules, set off by curly braces, and the attachment of additional parameters to non-terminals. These features raise the formal power of DCGs well beyond context free, in fact to the top of the Chomsky hierarchy. Moreover, with the recursive-descent parser inherent in Prolog, it is the responsibility of the programmer to write efficient, terminating rule sets. This is offset by the advantages Prolog offers in rapid prototyping and the ability easily to define new syntaxes and metalanguages especially tailored to a particular domain. The author’s syntactic pattern-recognition system for biological sequence data, called GenLang, is a case in point (Searls and Dong, 1993).

GenLang was designed to process efficiently the huge input strings of DNA sequence data currently available and being produced at an exponentially increasing rate. Instead of
Prolog's linked lists, the input is represented in a 'C' array for efficiency which, however, entails extra bookkeeping behind the scenes in the parser; in fact, up to a dozen 'hidden' parameters are attached to non-terminals in order to manage information about the input string, the parse tree and the cost of imperfect matching.

Queries take the form <pattern>:<parse variable> ⇒ <input>, where the pattern generally contains the top-level non-terminal in the grammar, the parse variable is a logic variable to which a parse tree will be bound, and the input is a DNA string (or file containing such a string). One other novel infix operator is the gap, denoted by an ellipse, which simply consumes some length of input that may be either unbounded (...) or bounded by a minimum and maximum extent (e.g. 3...75). Otherwise, GenLang grammars appear very much like ordinary DCGs, except that most objects, such as terminals, non-terminals, and gaps, may have attached to them lists of attributes, of the form:

<object>:[<attribute>,...<attribute>]

where the attributes are generally operations on keywords, and are of four types: (i) control operators, which modify the course of the parse and the position on the input string (departing from 'pure' logic programming, usually for the sake of efficiency); (ii) constraint attributes, which impose limits on quantities denoted by keywords, such as the cost (normally, the number of mismatches) of a subtree; (iii) specification attributes, which redefine the values of keyword quantities with arbitrary expressions; and (iv) assignment attributes, which bind the values of keyword quantities to logic variables that may be carried through the parse and used to report information at the top level. Besides the control attributes which can modify the backtracking search, another set of features, using the prefix operator @, can control the position of the parse on the input string, allowing for arbitrary translocations and additional kinds of constraints. This syntax is demonstrated in the following (relatively complicated) GenLang rule:

orf: [once, cost=C-S/10,
   parse=[span, cost]] ⇒
   'atg',...:[step=3, S=(size>30)],
   @End, stop:[C=cost], @End.

Here, the essential framework of the rule states that the feature orf consists of the terminal string 'atg', followed by a gap, followed by the feature stop. The control attribute step=3 specifies that the gap is to increase in increments of three, and the constraint attribute size>30 indicates that the range 30 or less can be ignored. The latter is combined with an assignment that binds the value of size to the logic variable S, just as the C=cost assigns to C the cost of the stop. The cost of the rule, by default, is the sum of the costs of its components: the number of mismatches in the terminal strings plus (recursively) the costs of any non-terminals. Here, however, the specification attribute cost=C-S/10 redefines the cost of this rule to be an algebraic function of the size of the gap and the cost of the stop feature. Another specification attribute controls what information will appear in the parse tree; in this case, just the numerical span of the orf and its computed cost, without the detailed subtree that would ordinarily be displayed.

The control attribute keyword once indicates that this rule can only succeed once in any starting position, and in this case prevents backtracking into the interior gap. (The effect of this keyword is similar to that of the Prolog 'cut', but it is just one of a family of such controls on the expansion of the parse.) This insures that orfs always end at a stop codon, and never include one in frame. The @Ends in the rule body refer to position in the input string; the first one binds the position just before the stop to the variable End, and the second, since the variable is now bound, resets the input to that position. Thus, the span reported for the orf would extend up to, but not include, the stop codon. Again, there are a large number of variations on the @ control syntax that allow for arbitrary movement on the input.

Not only are gaps first-class objects in GenLang, but they are in some ways the most important objects in terms of the implementation. This is because gaps, which 'skip over' input, are the search engines for individual features of interest that they precede. Since gaps produce the majority of the non-determinism, or backtracking behavior, in a grammar, they not only must be made very efficient, but they are treated specially by the grammar compiler. Gaps are typically not executed immediately in the course of a parse, but rather are 'packaged' and passed down the parse tree to succeeding non-terminals, until they encounter some feature with which they may combine for more efficient evaluation. For example, the combination of such a 'lazy' gap with a terminal string can be more efficiently evaluated than by brute force matching à la ordinary DCGs. GenLang will, at the option of the user, create a hash table of the position of every substring of a given length in the input string, so that a gap/string combination can simply be looked up for immediate evaluation, instead of scanning.

Lazy gaps also permit greater variety in the search strategy, which in the logic-based parser is ordinarily breadth-first on the input, i.e. all applicable rules are tried at every position before moving on to the next position. A lazy gap, however, is passed to the first applicable rule, which can be tried in every possible position allowed by the gap, before passing the gap to the next alternative clause or rule. This describes depth-first search on the input. The search strategy in GenLang can be varied locally through the use of the rule attributes deep, wide, or best.
In order more easily to specify some of the linguistically complex features expected in this domain, the author has developed and characterized a formalism called string variable grammar (SVG) (Searls, 1988, 1995a), as another augmentation to DCGs. SVGs allow logic variables as objects in the grammar, standing for uninstantiated terminal strings. It is thus possible to specify such features economically as:

\[
\begin{align*}
\text{tandem-repeat} & \rightarrow X, X. \\
\text{inverted-repeat} & \rightarrow X, \ldots, -X. \\
\text{pseudoknot} & \rightarrow X, \ldots, Y, -X, \ldots, -Y.
\end{align*}
\]

where the tilde is a prefix operator denoting reverse complement. Like gaps, string variables can assume any length, though in practice they are generally constrained by the appropriate attributes. (A gap, in fact, can be seen as simply an ‘anonymous’ string variable.) In the inverted repeat rule above, the string variables constitute the two base-paired halves of the stem, while the interior gap represents the unpaired loop.

Note that a pseudoknot is a kind of interleaved inverted repeat, where half of the stem of one stem-and-loop resides within the loop of a second stem-and-loop. A number of stem-and-loop structures are illustrated in Figures 2 and 3, and the latter also contains a pseudoknot. To specify these and many related features (which are often beyond context free) using ordinary DCGs requires much more complex rules (Searls, 1989).

As has been noted, the cost of a feature is by default the number of mismatches in terminal strings eventually contained within the feature, though this can be arbitrarily redefined. A number of other cost functions are available as built-ins, including weight matrices (with a variety of evaluation methods), edit distance and a facility for substitution matrices that allow for individualized costs for every possible alphabetic substitution. Costs combine and propagate up the parse tree until they are constrained and/or redefined by the appropriate attributes, so that the entire parse tree can have its ‘stringency’ adjusted both globally and locally over subtrees.

A number of GenLang grammars have been described previously; here, results derived from several of them will be reviewed. The characteristic cloverleaf secondary structure of transfer RNA (tRNA), for example, is easily described by an SVG (Searls, 1989). A grammar originally designed for relatively simple bacterial tRNAs (Searls and Liebowitz, 1990) was enhanced so as to handle a greater degree of variation, including a specific type of intron, and to detect conserved regions in the primary (lexical) structure via weight matrices, in addition to the secondary structure. The resulting grammar, when used to parse the GenBank database...
of all known DNA sequences, was able to find >95% of known tRNAs among bacteria, invertebrates and primates (a total of >700) with virtually no false positives (Sears and Dong, 1993). This compared favorably with hard-coded procedural tRNA finders developed by others, and even achieved comparable efficiency. Figure 2 shows the results of running this grammar against the entire length of yeast chromosome III.

Another, more complex higher-order structure found in certain organisms is that of the group I intron. This is a form of self-splicing intron which is characterized, again, by both recurring primary sequence features and by a distinctive secondary structure, this time involving a pseudoknot (Cech, 1988). These are handled uniformly by the SVG formalism, as shown above. In actual use, this significantly more complex grammar was able to find the majority of such features in the genome of a fungal mitochondrion, together with a variety of other features which are typically pointed out in such sequences when they are described by biologists. These results, illustrated in Figure 3, suggest the utility of grammars both as declarative descriptions of higher-order structures in a consistent, well-founded syntax, and as a means of assisting in the annotation of sequence data via parsing. Others have created systems capable of recognizing tRNAs, group I introns, and structures of similar complexity, using pattern-recognition techniques that are also syntactic in flavor, though not explicitly based on grammar formalisms (Gautheret et al., 1990).

Besides encouraging formal declarative descriptions of the many complex structures recognized in sequence data, the use of logic grammars for their procedural interpretation offers many advantages. Prolog is an outstanding rapid prototyping language, which carries over into grammar design, and it is also easy to write compilers and design metalanguages for abstract interpretation of grammars with domain-specific syntactic features and speed-ups. The ability to embed in-line code in grammar rules, including foreign code or even entire expert systems, means that the grammar can be used as a framework to house special-purpose algorithms and apply them selectively to regions of interest determined by a higher structural context. Particularly for computationally expensive algorithms, such focus can lead to considerable savings. Moreover, the ease of passing parameters up and down the parse tree means that these algorithms, which are often highly parameterized, can be fine-tuned to suit the context; also, the results can be synthesized and combined under the direction of the housing grammar framework.

These virtues of the linguistic approach to sequence analysis meet their most stringent test in the problem of detecting genes in raw sequence data, as described in the next section.

**Syntactic gene finding**

With the development of large-scale DNA sequencing technology, more and more long stretches of uncharacterized DNA are becoming available, and a natural question is to ask where the genes are in such sequences. Since genes represent only a few percent of the DNA of the genome, answering the question is not straightforward, and this has stimulated the development of increasingly effective gene-finding algorithms.

For some time, the accepted way of approaching this question was simply to attempt to detect protein-coding regions, which may be expected to have certain characteristics that distinguish them from non-coding regions. Finding
long open reading frames is the first heuristic in searching for coding sequences, but it does not suffice. In vertebrates, the average length of the coding exons (~130 bases) is not sufficient reliably to distinguish them from open reading frames that could be expected to exist in extensive non-coding regions by random chance. Codon usage statistics can be employed to evaluate a putative coding sequence for its tendency to use preferred triplets, rather than just any triplet that is not a stop codon, and this approach has been extended to longer 'words', with hextuple frequencies being an oft-used heuristic to distinguish coding from non-coding sequence.

More subtle tendencies can also be exploited; for example, the early TESTCODE algorithm (Fickett, 1982) made use of an observed tendency in coding regions for bases to recur in the same relative position with a cyclicity of length three, i.e. with respect to the reading frame. Such statistical tests produce a 'score' indicating coding potential within a window on the input sequence, with the reliability of the score increasing with window size. Unfortunately, once again the average size of vertebrate exons is generally smaller than the optimum window size, and by and large these so-called compositional methods do not offer anything approaching...
absolute reliability (Fickett and Tung, 1992). Nevertheless, compositional methods have become increasingly sophisticated, and suffice to find significant portions of a number of exons out of the many that may constitute a gene, thus drawing attention to the appropriate region. Moreover, approaches such as GRAIL have made use of the powerful technique of combining evidence from several compositional methods by way of a neural net (Uberbacher and Mural, 1991).

The compositional approach may be contrasted with the strictly syntactic or rule-based approach to gene recognition. The author showed that the basic "rules" of gene expression, including splicing and even flanking control regions, can be conveniently encoded in a grammar that is able to recognize DNA sequences capable of producing proteins (Searls, 1987; Searls and Noordewier, 1991). This approach had great appeal because of the known virtues of declarative, hierarchical specifications. However, just as syntax underspecifies natural language, allowing grammatically correct yet nonsensical sentences, so does a gene syntax underspecify the valid protein-encoding genes in a genome. Apart from the deterministic machinery of protein translation, such a grammar depends upon reliable recognizers of specific features in the DNA, such as start sites and splice junctions. Identifying such sites—the word recognition problem described above—turns out to be at least as dicey as finding coding sequence. This key distinction, between syntactic and compositional methods, was recognized early and termed "gene search by signal" versus "gene search by content" (Staden, 1984).

In fact, the problem of syntactic recognition of genes may be thought of as the dual of the problem of compositional recognition of coding regions, i.e. any comprehensive solution to one would effectively constitute a solution to the other. In other words, if we could invariably identify the signals for where genes, transcripts, splice sites, etc., occur, we would know the coding regions, and if we could recognize coding sequence with exactitude, we could easily infer the structure of the genes. It soon became evident that neither approach in isolation would suffice, and the most successful programs today are "hybrid" systems that combine compositional evidence with rule-based frameworks that assemble coding regions into plausible gene structures. Thus, the experience in this domain can be seen to have paralleled that in fields such as character recognition and speech processing, where hybrid statistical and syntactic pattern-recognition systems have proven most effective (Searls and Taylor, 1993).

A GenLang gene grammar was developed in which conventional weight matrices were used to detect signal words for translation and processing, and in some cases promoter signals for transcription (see Figure 4). In addition to syntactic and size constraints, compositional measures contributed to the cost both locally within putative exons (based on hextuple frequencies for coding versus non-coding regions) and globally for entire exons. As before, sequences were parsed left to right and a parse tree built, which also provided a structure by which both signal and compositional costs could be combined, with weights and thresholds determined empirically. Thus, signals were only searched for in contexts of the developing parse where they "made sense".

Other rule-based architectures had also incorporated compositional measures into frameworks for assembling genes (Fields and Soderlund, 1990; Gelfand, 1990; Guigo et al., 1992). A problem common to such gene prediction programs is that of the combinatorics of exon assembly: it is not uncommon for genes to have more than a dozen exons, and since exons cannot be delineated with absolute certainty, an exponential number of combinations may have to be postulated and evaluated. Besides careful adjustment of thresholds for exon prediction from compositional evidence, a variety of techniques can and have been used to ameliorate the computational expense of assembly. By far the most successful approach to this problem has been the adoption of dynamic programming algorithms (Gelfand and Roytberg, 1993; Snyder and Stormo, 1993). These techniques, in theory, allow all possible locations of exons to be evaluated in the context of an entire gene architecture, and in polynomial time. Although parsing of context-free languages can be performed with similar efficiency, the problem of examining all parses, where there may be an exponential number of them, may be intractable even so.

In GenLang, the technique of chart parsing (a linguistic version of dynamic programming) was used to achieve reasonable efficiency, and accuracy was comparable to the best results at the time (Dong and Searls, 1994). However, ever more efficient and accurate systems are now outpacing what is likely to be possible with a general-purpose parser (Burset and Guigo, 1996; Fickett, 1996). Moreover, the function of the gene grammar in this case is primarily to serve as a framework on which to organize compositional sensors, which in all provide the most effective cues. What, then, is the future of syntactic models of gene structure?

One answer may depend on the accumulation of a more complete catalog of genes leading to a richer set of themes and variations (for which grammars are particularly well suited). For example, compositional methods are known to perform better on some classes of genes than others (perhaps partly because of skewed training sets), and they have very little to say about certain specialized gene structures, such as those associated with the immune system which undergo distinctive types of rearrangement in the genome in advance of gene expression. Such specialized forms of genes, gene regions, and organized gene families may prove to be more the rule than the exception, as a more global view of genome structure emerges. The same may be true of mechanisms of gene expression such as splicing. Another increasingly important theme is that of alternative forms of transcription...
for the same gene, e.g. multiple start and end sites, and especially alternative splice sites, which add layers of combinatoric variation well suited to grammatical expression (Searls and Noordewier, 1991). All this, and the increasingly elaborate schemata of gene regulation currently being elucidated, will inevitably lead to more detailed and probably more hierarchical models that are the natural domain of syntactic description and analysis. As shown in Figure 5, such models have potential for extending all the way from nucleotide-level signals to global organization of gene regions.

The notion of modeling gene structure and the mechanisms of gene expression, in fact, is the most appealing aspect of the syntactic approach. Compositional methods (like neural nets in general) may be thought of as capturing strictly empirical, post hoc, statistical correlations, with little or no rational model involved beyond some initial, crude structuring of the problem. Of course, parsers such as GenLang may also be employed strictly as opportunistic detectors using ad hoc rules without any mechanistic forethought, but a grammar at least provides an opportunity for rational design correlated
grammars and parsers can be seen to model biological
structure (Searls, 1992). This is even more evident in the
intriguing possibility of a compositional semantics of
functional motifs of proteins, which could theoretically
assemble the ‘meaning’, qua function, of the protein as a
whole.

A subtle counterargument to this view holds that current
gene grammars tend inappropriately to combine syntactic
elements that in fact come into play at different times in
different compartments of the cell. Why, for example, should
a gene finder that purports to model gene expression take
advantage of knowledge of open reading frames (a function
of the translational apparatus) when evaluating the likely
location of splice sites (determined during RNA processing,
long before translation), and vice versa? The author has, in
fact, suggested that separate such grammars could be
combined for the purposes of detection, with the understanding
that they are imperfect and incomplete in isolation, but syner-
gistic in their interaction. [This, in fact, has interesting formal
consequences, entailing as it does the intersection of
languages (Searls, 1992).] Moreover, it may eventually prove to be
presumptuous to exclude any information from such models;
recent evidence suggests that, in fact, reading frame does
have an effect on splicing, through as yet unknown mechan-
isms (Naegeler, 1992). Clearly, it will be necessary to maintain
an intelligent balance between the use of grammars to model
current knowledge and the need to shape them in new
directions that seem to produce better results in parsing,
possibly leading to new or altered hypotheses in the process.

Acknowledgement
This work was supported by the Department of Energy, Office of Energy
Research, under grant number ER61371.

References
Bentolila, S. (1996) A grammar describing ‘biological binding operators’ to
Brendel, V. and Busse, H. G. (1984) Genome structure described by formal

\[\text{immunoglobulin\_superfamily\_gene\_region} \rightarrow v\_genes(S), \ldots, d\_segments(S), \]
\[(\text{recombinable}(S,S_1,S_2)), \ldots, j\_segments(S_2), \]
\[\ldots, \text{transcriptional\_enhancer}, \ldots, c\_segments.\]

\[v\_genes(S) \rightarrow [] | \]
\[v\_gene, \text{recombination\_signal}(S), \ldots, v\_genes(S).\]

\[v\_gene \rightarrow \text{promoter, \ldots, leader\_peptide, intron, v\_segment.}\]

\[\text{recombination\_signal}([H,G,N]) \rightarrow \]
\['\text{cacagtg}':(\text{cost<}2, H=\text{sequence}), 12...23:G=\text{size}, \]
\['\text{acaaaaacc}':(\text{cost<}2, N=\text{sequence}).\]

Fig. 5. A fragment of a possible immunoglobulin superfamily gene grammar.
Unlike the ‘vanilla’ gene grammar used for Figure 4, this specialized
grammar would capture the specific architecture of this family. The cross-
section of rules illustrated span from the level of genome organization down
to specialized signals. While the lower-level rules might be useful in
recognition, the major utility of such grammars might be in the organization
of knowledge about genetic organization in computer-readable form.
with known biological structures and phenomena. Indeed,
one of the motivations for the use of generative grammars in
natural language analysis is the hope that forcing observa-
tions to conform to concise declarative formulations will lead to
clarifying generalizations, almost as a by-product; the
extreme version of this view is that the grammars discovered
may in fact reflect actual cognitive mechanisms in some
manner. The author has described a number of ways in which
grammars and parsers can be seen to model biological
mechanisms and structures (Searls, 1993); as noted above,
for example, parse trees for secondary structural grammars
physically resemble those structures, even to the extent that
ambiguity in the grammars reflects alternative secondary
structure (Searls, 1992).

It may be argued that the necessity of including composi-
tional data to detect genes reliably compromises the validity
of the syntactic approach. In fact, the situation is no different
than for natural language, where the set of semantically
meaningful sentences is a small subset of the grammatically
acceptable sentences. Grammar may be thought of as a
framework, reflecting some core mechanism of gene expres-
sion to which protein ‘utterances’ are bound to conform, but
in addition to which they are subject to biochemical and
thermodynamic context and, more broadly, evolutionary
selection. In the terminology of Chomsky, a gene grammar
should reflect the competence of the cell to deal with the DNA
sequence, and not necessarily its actual performance. No
doubt, current grammars are inadequate as models of the
cellular machinery, and grammars may or may not prove to be
suitable receptacles for such knowledge as it accumulates, but
they appear promising for the moment.

For example, current thinking on the mechanism of gene
regulation (e.g. Tjian, 1995) and splicing (Niwa et al., 1992)
points toward multiple interacting factors such as (in the case
of splicing) the combined ‘strengths’ of several recognition
sites, the lengths of flanking exons and/or introns, local
conformation of the RNA, and other characteristics of the
sequence neighborhood. Not only do grammars intrinsically
model such hierarchical combinations of factors, but they
provide a framework for the formal evaluation of the
semantics of the whole in terms of operations on its parts,
where those operations are functionally mapped from strictly
syntactic combinations of symbols. Such functional mappings
may be as simple as combinatoric ‘truth-functionals’
determining the state of some binary genetic switch, or as
sophisticated as cooperative thermodynamic binding equa-
tions, but the encouraging sign is the emerging theme of the
whole being determined by combined characteristics of the
parts. This is even more evident in the intriguing possibility of
a compositional semantics of functional motifs of proteins,
which could theoretically assemble the ‘meaning’, qua func-
tion, of the protein as a whole.

For example, current thinking on the mechanism of gene
regulation (e.g. Tjian, 1995) and splicing (Niwa et al., 1992)
points toward multiple interacting factors such as (in the case
of splicing) the combined ‘strengths’ of several recognition
sites, the lengths of flanking exons and/or introns, local
conformation of the RNA, and other characteristics of the
sequence neighborhood. Not only do grammars intrinsically
model such hierarchical combinations of factors, but they
provide a framework for the formal evaluation of the
semantics of the whole in terms of operations on its parts,
where those operations are functionally mapped from strictly
syntactic combinations of symbols. Such functional mappings
may be as simple as combinatoric ‘truth-functionals’
determining the state of some binary genetic switch, or as
sophisticated as cooperative thermodynamic binding equa-
tions, but the encouraging sign is the emerging theme of the
whole being determined by combined characteristics of the
parts. This is even more evident in the intriguing possibility of
a compositional semantics of functional motifs of proteins,
which could theoretically assemble the ‘meaning’, qua func-
tion, of the protein as a whole.

A subtle counterargument to this view holds that current
gene grammars tend inappropriately to combine syntactic
elements that in fact come into play at different times in
different compartments of the cell. Why, for example, should
a gene finder that purports to model gene expression take
advantage of knowledge of open reading frames (a function
of the translational apparatus) when evaluating the likely
location of splice sites (determined during RNA processing,
long before translation), and vice versa? The author has, in
fact, suggested that separate such grammars could be
combined for the purposes of detection, with the understanding
that they are imperfect and incomplete in isolation, but syner-
gistic in their interaction. [This, in fact, has interesting formal
consequences, entailing as it does the intersection of
languages (Searls, 1992).] Moreover, it may eventually prove to be
presumptuous to exclude any information from such models;
recent evidence suggests that, in fact, reading frame does
have an effect on splicing, through as yet unknown mechan-
isms (Naegeler, 1992). Clearly, it will be necessary to maintain
an intelligent balance between the use of grammars to model
current knowledge and the need to shape them in new
directions that seem to produce better results in parsing,
possibly leading to new or altered hypotheses in the process.

Acknowledgement
This work was supported by the Department of Energy, Office of Energy
Research, under grant number ER61371.

References
Bentolila, S. (1996) A grammar describing ‘biological binding operators’ to
Brendel, V. and Busse, H. G. (1984) Genome structure described by formal