CHAPTER

2

The Computational Linguistics of Biological Sequences

David B. Searls

1 Introduction

Shortly after Watson and Crick's discovery of the structure of DNA, and at about the same time that the genetic code and the essential facts of gene expression were being elucidated, the field of linguistics was being similarly revolutionized by the work of Noam Chomsky [Chomsky, 1955, 1957, 1959, 1963, 1965]. Observing that a seemingly infinite variety of language was available to individual human beings based on clearly finite resources and experience, he proposed a formal representation of the rules or *syntax* of language, called generative grammar, that could provide finite—indeed, concise—characterizations of such infinite languages. Just as the breakthroughs in molecular biology in that era served to anchor genetic concepts in physical structures and opened up entirely novel experimental paradigms, so did Chomsky's insight serve to energize the field of linguistics, with putative correlates of cognitive processes that could for the first time be reasoned about

axiomatically as well as phenomenologically. While Chomsky and his followers built extensively upon this foundation in the field of linguistics, generative grammars were also soon integrated into the framework of the theory of computation, and in addition now form the basis for efforts of computational linguists to automate the processing and understanding of human language.

Since it is quite commonly asserted that DNA is a richly-expressive *language* for specifying the structures and processes of life, also with the potential for a seemingly infinite variety, it is surprising that relatively little has been done to apply to biological sequences the extensive results and methods developed over the intervening decades in the field of formal language theory. While such an approach has been proposed [Brendel and Busse, 1984], most investigations along these lines have used grammar formalisms as tools for what are essentially information-theoretic studies [Ebeling and Jimenez-Montano, 1980; Jimenez-Montano, 1984], or have involved statistical analyses at the level of vocabularies (reflecting a more traditional notion of comparative linguistics) [Brendel et al., 1986; Pevzner et al., 1989a,b; Pietrokovski et al., 1990]. Only very recently have generative grammars for their own sake been viewed as models of biological phenomena such as gene regulation [Collado-Vides, 1989a,b, 1991a], gene structure and expression [Searls, 1988], recombination [Head, 1987] and other forms of mutation and rearrangement [Searls, 1989a], conformation of macromolecules [Searls, 1989a], and in particular as the basis for computational analysis of sequence data [Searls, 1989b; Searls and Liebowitz, 1990; Searls and Noordewier, 1991].

Nevertheless, there is an increasing trend throughout the field of computational biology toward abstracted, hierarchical views of biological sequences, which is very much in the spirit of computational linguistics. At the same time, there has been a proliferation of software to perform various kinds of pattern-matching search and other forms of analysis, which could well benefit from the formal underpinnings that language theory offers to such enterprises. With the advent of very large scale sequencing projects, and the resulting flood of sequence data, such a foundation may in fact prove essential.

This article is intended as a prolegomenon to a formally-based computational linguistics of biological sequences, presenting an introduction to the field of mathematical linguistics and its applications, and reviewing and extending some basic results regarding structural and functional phenomena in DNA and protein sequences. Implementation schemes will also be offered, largely deriving from logic grammar formalisms, with a view toward practical tools for sequence analysis.

2 Formal Language Theory

This section will provide a compact but reasonably complete introduction to the major results of formal language theory, that should allow for a basic

understanding of the subsequent sections by those with no background in mathematical linguistics. Proofs will be omitted in this section; some will be offered later as regards biological sequences, and will use a range of proof techniques sufficient to demonstrate the basic methodologies of the field, but by and large these will be simple and by mathematical standards "semi-formal." Readers interested in further studies along these lines are encouraged to consult textbooks such as [Sudkamp, 1988; Hopcroft and Ullman, 1979; Harrison, 1978] (in order of increasing difficulty). Those already familiar with the subject area should skip this section.

2.1 The Formal Specification of Languages

Formally, a *language* is simply a set of *strings* of characters drawn from some *alphabet*, where the alphabet is a set of symbols usually denoted by Σ. One such language would be simply the set of *all* strings over an alphabet $\Sigma = \{0,1\}$; this "maximal" language is indicated by the use of an asterisk, e.g.

$$\Sigma^* = \{0,1\}^* = \{\, \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \cdots \,\} \tag{1}$$

Here, the ε represents the *empty string* or string of length zero; the set containing ε, however, should not be confused with the empty set $\varnothing$. The challenge of computational linguistics is to find concise ways of specifying a given (possibly infinite) language $L \subseteq \Sigma^*$, preferably in a way that reflects some underlying model of the "source" of that language. We can use informal descriptions that make use of natural language, such as in the following example:

$$L_a = \{\, w \in \{0,1\}^* \mid w \text{ begins with a 0 and contains at least one 1} \,\} \tag{2}$$

(The vertical bar notation is used to define a set in terms of its properties; this specification would be read "the set of all strings *w* of 0's and 1's *such that* each *w* begins with a 0 and . . .") However, properties expressed in natural language are typically neither precise enough to allow for easy mathematical analysis, nor in a form that invites the use of computational support in dealing with them. On the other hand, simply exhaustively enumerating languages such as the example in (2) is also clearly ineffective—in fact, impossible:

$$L_a = \{\, 01, 001, 010, 011, 0001, 0010, 0011, 0100, \cdots \,\} \tag{3}$$

The remainder of this section will examine formal methods that have been used to provide finite specifications of such languages.

2.1.1 Regular Expressions and Languages. A widely-used method of specifying languages is by way of *regular expressions*, which in their mathematically pure form use only three basic operations. These operations are given below, using a notation in which a regular expression is given in bold type, and the language "generated" by that expression is derived by the ap-

plication of a function L (defined recursively in the obvious way):

(i) **concatenation**, denoted by an infix operator '$\cdot$', or more often by simply writing symbols in juxtaposition, e.g. $L(\mathbf{01}) = \{01\}$;

(ii) **disjunction** (or logical OR), denoted in this case by the infix operator '+', e.g. $L(\mathbf{0+1}) = \{0,1\}$; and

(iii) **Kleene star**, denoted by a postfix superscript '*****', represents the set containing zero or more concatenated instances of its argument, e.g. $L(\mathbf{0^*}) = \{ \varepsilon, 0, 00, 000, 0000, \cdots \}$.

The latter operation is also known as the *closure* of concatenation. Note the connection between the definition of Kleene star and our previous use of the asterisk:

$$\Sigma^* = L(\mathbf{(0+1)^*}) \qquad \text{for } \Sigma = \{0,1\} \tag{4}$$

One additional non-primitive operator, a postfix superscript '$^+$', is used to specify one or more occurrences of its argument. This is the *positive closure* of concatenation, defined in terms of concatenation and Kleene star as

$$L(\mathbf{0^+}) = L(\mathbf{00^*}) = \{ 0, 00, 000, 0000, \cdots \} \tag{5}$$

The language from our running example of (2) can now be described using any of several regular expressions, including

$$L_a = L(\mathbf{00^*1(0+1)^*}) \tag{6}$$

From this point, we will dispense with the $L(\ )$ notation and let the regular expression standing alone denote the corresponding language. Any such language, that can be described by a regular expression, will be called a *regular language* $(\mathrm{RL})^*$.

2.1.2 Grammars. Such regular expressions have not only found wide use in various kinds of simple search operations, but are also still the mainstay of many biological sequence search programs. However, it is a fact that many important languages simply cannot be specified as regular expressions, e.g.

$$\{ 0^n 1^n \mid n \geq 1 \} \tag{7}$$

where the superscript integers denote that number of concatenated symbols, so that (7) is the set of all strings beginning with any non-zero number of 0's followed by an equal number of 1's. This shortcoming of regular expressions for language specification can be remedied through the use of more powerful representations, called *grammars*. Besides a finite alphabet Σ of *terminal* symbols, grammars have a finite set of "temporary" *nonterminal* symbols (including a special *start* symbol, typically *S*), and a finite set of *rules* or *productions*; the latter use an infix '$\rightarrow$' notation to specify how strings containing nonterminals may be rewritten by expanding those embedded nontermi-

nals (given on the left-hand side of the arrow) to new substrings (given on the right-hand side). For instance, a grammar specifying the language L_a of (2) can be written:

$$\begin{array}{ll} S \rightarrow 0A & B \rightarrow 0B \\ A \rightarrow 0A & B \rightarrow 1B \\ A \rightarrow 1B & B \rightarrow \varepsilon \end{array} \tag{8}$$

Note that nonterminals are traditionally designated by capital letters. A *derivation*, denoted by an infix '$\Rightarrow$', is a rewriting of a string using the rules of the grammar. By a series of derivations from S to strings containing only terminals, an element of the language is specified, e.g.

$$S \Rightarrow 0A \Rightarrow 00A \Rightarrow 001B \Rightarrow 0010B \Rightarrow 00101B \Rightarrow 00101 \tag{9}$$

Often there will be multiple nonterminals in a string being derived, and so there will be a choice as to which nonterminal to expand; when we choose the leftmost nonterminal in every case, we say that the series is a *leftmost derivation.*

2.1.3 Context-Free Languages. Grammars such as that of (8), whose rules have only single nonterminals on their left-hand sides, are called *context-free*. The corresponding languages, i.e. those that can be generated by any such grammar, are called *context-free languages* (CFLs); it happens that they include the RLs and much more. For example, the language of (7) is specified by a grammar containing the following productions:

$$S \rightarrow 0A \qquad A \rightarrow 0A1 \qquad A \rightarrow 1 \tag{10}$$

Many other grammars can be used to describe this language, but no regular expression suffices. Another classic context-free (and not regular) language is that of *palindromes*, which in this case refer to "true" palindromes—strings that read the same forward and backward—rather than the biological use of this word to describe dyad symmetry (see section 2.4.1). We can denote such a language (for the case of even-length strings over any alphabet) as

$$\left\{ ww^R \mid w \in \Sigma^* \right\} \tag{11}$$

for any given Σ, where the superscript ^{R} denotes reversal of its immediately preceding string argument. Such languages can be specified by context-free grammars like the following, for $\Sigma = \{0,1\}$:

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon \tag{12}$$

(Note the use of the vertical bar to more economically denote rule disjunction, i.e. multiple rules with the same left-hand side.) Thus, context-free grammars are said to be "more powerful" than regular expressions—that is, the RLs are a proper subset of the CFLs.

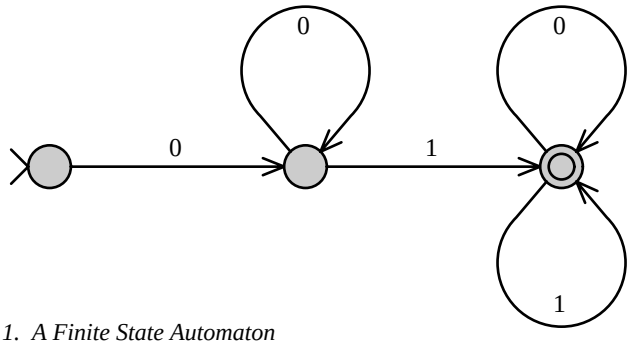

Figure 1. A Finite State Automaton

2.1.4 Automata. Grammars are intimately related to conceptual *machines* or *automata* which can serve as language recognizers or generators. For example, regular languages are recognized and generated by *finite state automata* (FSAs), which are represented as simple directed graphs, with distinguished starting and final nodes, and directed arcs labelled with terminal symbols that are consumed (if the machine is acting as a recognizer) or produced (if the machine is being used to generate a language) as the arc is traversed.

Figure 1 shows an FSA which again expresses the language L_a of (2). The starting node is at the left, and a final node is at the right. It can be seen that it corresponds closely to the "operation" of the regular expression given in (6). In fact, any such regular expression can be expressed as an FSA with a finite number of nodes or *states*, and vice versa, so that the languages recognized by FSAs correspond exactly to the regular languages.

More sophisticated machines are associated with more powerful languages. For example, by adding a limited memory capability in the form of a *stack* or simple pushdown store, we can create *pushdown automata* (PDA) that recognize context-free languages. Figure 2 shows a PDA which recognizes the language of (7). After consuming a 0, the machine enters a loop in which it pushes some symbol (typically drawn from a separate alphabet) on the stack for each additional 0 it consumes. As soon as a 1 is recognized, it makes a transition to another state in which those symbols are popped off the stack as each additional 1 is consumed. The stack is required to be empty in a final state, guaranteeing equal numbers of 0's and 1's. (As before, it is instructive to note how the PDA compares to the grammar of (10).) Once again, it can be shown that PDAs recognize all and only the CFLs.

More elaborate memory schemes can certainly be used in such machines, leading us to ask whether they can be made to recognize additional languages,

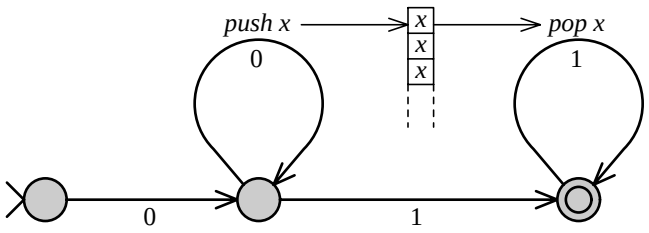

Figure 2. A Pushdown Automaton

and whether there are correspondingly more powerful grammar formalisms.

2.1.5 Context-Sensitive Languages. Indeed, even CFLs do not include some apparently simple languages, such as:

$$\{\, 0^n 1^n 2^n \mid n \geq 1 \,\} \qquad \{\, 0^i 1^j 2^i 3^j \mid i,j \geq 1 \,\} \tag{13}$$

Note the similarity of these languages to (7), which *is* a CFL. We can intuit why a PDA could not recognize (13a), for instance, by noting that the stack would have to be emptied in the course of recognizing the string of 1's, leaving no way to "count" the 2's. Another well-known class of non-context-free languages are the *copy languages*:

$$\{\, ww \mid w \in \Sigma^* \,\} \tag{14}$$

However, by relaxing our restriction and allowing more than one symbol on the left-hand sides of grammar rules (but always including at least one nonterminal), all these languages are encompassed. Such a grammar will be called *context-sensitive* if the left-hand side of each rule is not longer than its right-hand side. Note that this effectively excludes languages containing ε (such as (14)), since any rule deriving ε would necessarily have a right-hand side shorter than its left-hand side; we will often supplement the languages specified by such grammars by allowing ε, for purposes of comparison. The corresponding context-sensitive languages (CSLs), augmented where necessary with ε, properly contain the CFLs as well as the examples in (13) and (14). For instance, a grammar specifying (13a) is as follows:

$$\begin{array}{lll} S \to 0SBC & 0B \to 01 & CB \to BC \\ S \to 0BC & 1B \to 11 & C \to 2 \end{array} \tag{15}$$

This grammar specifies (13a) via sequences of derivations like the following. Note how in the second line the context-sensitive rule allows *B*'s to traverse *C*'s leftward to their final destinations:

$$\begin{aligned} S &\Rightarrow 0SBC \Rightarrow 00SBCBC \Rightarrow 000BCBCBC \Rightarrow 0001CBCBC \\ &\Rightarrow 0001BCCBC \Rightarrow 00011CCBC \Rightarrow 00011CBCC \Rightarrow 00011BCCC \\ &\Rightarrow 000111CCC \Rightarrow 0001112CC \Rightarrow 00011122C \Rightarrow 000111222 \end{aligned} \tag{16}$$

The machines associated with CSLs are called *linear-bounded automata*, which can move in *either* direction on the input, and whose memory scheme consists of the ability to overwrite symbols on the input (but not beyond it). The requirement that each rule's right-hand side be at least as long as its left-hand side ensures that strings produced by successive derivations never grow longer than the final terminal string, and thus exceed the memory available to the automaton.

2.2 The Chomsky Hierarchy and Subdivisions

If there is no constraint on the number of symbols on the left hand sides of rules, the grammar is called *unrestricted*, and the corresponding languages, called *recursively enumerable*, contain the CSLs and much more. The automaton corresponding to recursively enumerable languages is, in fact, the Turing machine itself, which establishes an important link to general algorithmic programming.

This completes the basic *Chomsky hierarchy* of language families, which are related by set inclusion (ignoring ε) as follows:

$$\text{regular} \subset \text{context-free} \subset \text{context-sensitive} \subset \text{unrestricted} \tag{17}$$

Care must be taken in interpreting these set inclusions. While the *set* of RLs is a subset of the CFLs, since any RL can be expressed with a context-free grammar, it is also the case that any CFL (or indeed any language at all) is a subset of an RL, namely Σ^*. That is, by ascending the hierarchy we are augmenting the range of languages we can express by actually *constraining* Σ^* in an ever wider variety of ways.

The Chomsky hierarchy has been subdivided and otherwise elaborated upon in many ways. A few of the important distinctions will be described.

2.2.1 Linear Languages. Within the CFLs, we can distinguish the *linear* CFLs, which include examples (7) and (11) given above. The linear CFLs are those that can be expressed by grammars that never spawn more than one nonterminal, i.e. those in which every rule is of the form

$$A \to uBv \qquad \text{or} \qquad A \to w \tag{18}$$

where A and B are any nonterminal and $u,v,w \in \Sigma^*$. The machines corresponding to linear CFLs are *one-turn PDAs*, which are restricted so that in effect nothing can be pushed on the stack once anything has been popped.

If either u or v is always empty in each rule of the form (18a), the resulting grammars and languages are called *left-linear* or *right-linear*, respectively, and the corresponding languages are RLs. For example, the language L_a

of (2), which was first specified by regular expressions and is thus regular, is also described by the right-linear grammar of (8). In one sense, then, the linear CFLs can be thought of as the simplest CFLs that are not regular.

2.2.2 Deterministic and Nondeterministic Languages. Other classifications depend on the nature of derivations and the behavior of the automata that produce or recognize languages. One such distinction is that between *deterministic* and *nondeterministic* languages and automata. Essentially, a deterministic automaton is one for which any acceptable input in any given state of the automaton will always uniquely determine the succeeding state. A deterministic language, in turn, is one that can be recognized by some deterministic machine. The FSA of Figure 1 is deterministic, since no node has more than one arc leaving it with the same label. In a nondeterministic FSA, there might be more than one arc labelled "0" leaving a node, for instance, and then a choice would have to be made; in attempting to recognize a given input, that choice might later prove to be the wrong one, in which case a recognizer would somehow have to backtrack and try the alternatives.

The PDA of Figure 2 is also deterministic, and thus the language of (7) is a deterministic CFL. This can be seen from the fact that the automaton merely has to read 0's until it encounters its first 1, at which point it begins popping its stack until it finishes; it need never "guess" where to make the switch. However, the palindromic language of (11) is a nondeterministic CFL, since the automaton has to guess whether it has encountered the center of the palindrome at any point, and can begin popping the stack.

Any nondeterministic FSA may be converted to a deterministic FSA, though obviously the same cannot be said of PDAs. Thus, the deterministic subset of CFLs properly contains the RLs.

2.2.3 Ambiguity. Another useful distinction within the CFLs concerns the notion of *ambiguity*. Formally, we say that a grammar is ambiguous if there is some string for which more than one leftmost derivation is possible. As it happens all of the example grammars we have given are unambiguous, but it is easy to specify ambiguous grammars, e.g.

$$S \rightarrow S0S \mid 1 \tag{19}$$

for which it can be seen that the string '10101' has two leftmost derivations:

$$\begin{aligned} &S \Rightarrow S0S \Rightarrow 10S \Rightarrow 10S0S \Rightarrow 1010S \Rightarrow 10101 \\ &S \Rightarrow S0S \Rightarrow S0S0S \Rightarrow 10S0S \Rightarrow 1010S \Rightarrow 10101 \end{aligned} \tag{20}$$

However, the language specified by this grammar,

$$\{ (10)^n 1 \mid n \geq 0 \} \tag{21}$$

can in fact also be specified by a different grammar that is unambiguous:

$$S \rightarrow 10S \mid 1 \tag{22}$$

Can all languages be specified by some unambiguous grammar? The answer is no, and languages that cannot be generated by any such grammar are called *inherently ambiguous*. An example is the following CFL (not to be confused with the CSL of (13a)):

$$\{\, 0^i 1^j 2^k \mid i{=}j \text{ or } j{=}k, \text{ where } i,j,k \geq 1 \,\} \tag{23}$$

Intuitively, it can be seen that this language will contain strings, e.g. those for which $i{=}j{=}k$, that can be parsed in more than one way, satisfying one or the other of the grammar elements that impose constraints on the superscripts. Inherently ambiguous languages are necessarily nondeterministic; a PDA recognizing (23), for instance, would have to guess ahead of time whether to push and pop the stack on the 0's and 1's, respectively, or on the 1's and 2's.

2.2.4 Indexed Languages. The CSLs can also be subdivided. We choose only one such subdivision to illustrate, that of the *indexed languages* (ILs), which contain the CFLs and are in turn properly contained within the CSLs, except that ILs may contain ε. They are specified by indexed grammars, which can be viewed as context-free grammars augmented with indices drawn from a special set of symbols, strings of which can be appended to nonterminals (which we will indicate using superscripts). Rules will then be of the forms

$$A \rightarrow \alpha \qquad \text{or} \qquad A \rightarrow B^{\mathrm{x}} \qquad \text{or} \qquad A^{\mathrm{x}} \rightarrow \alpha \tag{24}$$

where α is any string of terminals and nonterminals, and x is a single index symbol. Now, whenever a rule of form (24a) is applied to expand a nonterminal A in the string being derived, all the indices currently attached to A in that input string are carried through and attached to each of the nonterminals (but not terminals) in α when it is inserted in place of A in the input string. For rules of form (24b), when A is expanded to B, x is added to the front of the string of indices on B in the terminal string being derived. Finally, for rules of form (24c), the index x at the head of the indices following A is removed, before the remainder of the indices on A are distributed over the nonterminals in α, as before.

This rather complicated arrangement may be clarified somewhat with an example. The following indexed grammar specifies the language of (13a):

$$\begin{array}{lll} S \rightarrow T^{\mathrm{s}} & A^{\mathrm{t}} \rightarrow 0A & A^{\mathrm{s}} \rightarrow 0 \\ T \rightarrow T^{\mathrm{t}} & B^{\mathrm{t}} \rightarrow 1B & B^{\mathrm{s}} \rightarrow 1 \\ T \rightarrow ABC & C^{\mathrm{t}} \rightarrow 2C & C^{\mathrm{s}} \rightarrow 2 \end{array} \tag{25}$$

Note that, but for the indices, this grammar is in a context-free form, though (13a) is not a CFL. Under this scheme, indices behave as if they were

on stacks attached to nonterminals, as may be seen in the following sample derivation (compare (16)):

$$\begin{aligned} S &\Rightarrow T^{\mathrm{s}} \Rightarrow T^{\mathrm{ts}} \Rightarrow T^{\mathrm{tts}} \Rightarrow A^{\mathrm{tts}}B^{\mathrm{tts}}C^{\mathrm{tts}} \Rightarrow 0A^{\mathrm{ts}}B^{\mathrm{tts}}C^{\mathrm{tts}} \\ &\Rightarrow 00A^{\mathrm{s}}B^{\mathrm{tts}}C^{\mathrm{tts}} \Rightarrow 000B^{\mathrm{tts}}C^{\mathrm{tts}} \Rightarrow 0001B^{\mathrm{ts}}C^{\mathrm{tts}} \Rightarrow 00011B^{\mathrm{s}}C^{\mathrm{tts}} \\ &\Rightarrow 000111C^{\mathrm{tts}} \Rightarrow 0001112C^{\mathrm{ts}} \Rightarrow 00011122C^{\mathrm{s}} \Rightarrow 000111222 \end{aligned} \tag{26}$$

Several types of machines are associated with ILs, including *nested stack automata*, whose name suggests a view of ILs as allowing stacks within stacks.

2.3 Lindenmayer Systems

Not all research in formal linguistics falls within the traditions of the Chomsky hierarchy and grammars in the form we have presented. One other important area will be described here, that of *Lindenmayer systems* or *L-systems.* These differ from the grammars above in that they have no nonterminals, and instead a derivation is accomplished by rewriting *every* terminal in a string simultaneously, according to production rules which of course have single terminals on the left and strings of terminals on the right. Actually, this describes the simplest, context-free form of these grammars, called a 0L-system, an example of which would be the following:

$$0 \rightarrow 1 \qquad\qquad 1 \rightarrow 01 \tag{27}$$

Beginning with a single 0, this produces a series of derivations as follows:

$$0 \Rightarrow 1 \Rightarrow 01 \Rightarrow 101 \Rightarrow 01101 \Rightarrow 10101101 \Rightarrow 0110110101101 \Rightarrow \cdots \tag{28}$$

The language of an L-system, called an *L-language*, is the set of all strings appearing in such a derivation chain. In this case, the language specified contains strings whose lengths are Fibonacci numbers, since in fact each string is simply the concatenation of the two previous strings in the series.

The 0L-languages, as it happens, are contained within the ILs, and thus within the CSLs (with ε), though they contain neither CFLs nor RLs in their entirety. Context-sensitive L-languages, on the other hand, contain the RLs but are only contained within the recursively enumerable languages [Prusinkiewicz and Hanan, 1989].

2.4 Properties of Language Families

Much of the content of formal language theory is concerned with examining the properties of families of languages—how they behave when various operations are performed on them, and what kinds of questions can be effectively answered about them. This section will give an overview of these properties.

2.4.1 Closure Properties. One such area of investigation is that of *closure* properties of families of languages, that is, whether applying various operations to languages leaves the resulting language at the same level in the Chomsky hierarchy. For example, all four of the language families in the hierarchy, and the ILs as well, are *closed under union*, which means that, for instance, the union of any CFL with any other CFL will always yield another CFL. Note, however, that the deterministic CFLS are not closed under union; consider the following two languages:

$$\{\, 0^i 1^j 2^j \mid i,j \geq 1 \} \qquad \{\, 0^i 1^i 2^j \mid i,j \geq 1 \} \tag{29}$$

Both these languages are deterministic, by reasoning similar to that given in a previous section for the language of (7). However, their union can be seen to be equivalent to the language of (23), which is inherently ambiguous and thus nondeterministic (though it is still a CFL).

The RLs, CFLs, ILs, CSLs, and recursively enumerable languages are all closed under concatenation (that is, the concatenation of each string in one language to each string in another, denoted $L_1 \cdot L_2$), as well as under the closures of concatenation (denoted L^* and L^+, the only difference being that the former contains ε whether or not L does). All are closed under intersection with any RL, e.g. the set of all strings occurring in both a given CFL and a given RL will always be a CFL. This fact will prove to be an important tool in proofs given below. CFLs, however, are not closed under intersection with each other, as can be seen from the fact that intersecting the two CFLs of (29) produces the CSL of (13a). The same is true of ILs, though CSLs and recursively enumerable languages *are* closed under intersection.

Another operation that will prove important in many proofs is that of *homomorphism*. A homomorphism in this case is a function mapping strings to strings, that is built upon a function mapping an alphabet to strings over a (possibly different) alphabet, by just transforming each element of a string, in place, by the latter function. For a function h on an alphabet Σ to extend to a homomorphism on strings over that alphabet, it is only necessary that it *preserve concatenation*, that is, that it satisfy

$$h(u) \cdot h(v) = h(uv) \quad \text{for } u,v \in \Sigma^*, \qquad \text{and} \qquad h(\varepsilon) = \varepsilon \tag{30}$$

For instance, given a homomorphism φ based on the functions φ(0)=ε, φ(1)=00, and φ(2)=φ(3)=1, we would have φ(123031200)=00111001. All four language families in the Chomsky hierarchy (and ILs as well) are closed under homomorphisms applied to each of the strings in a language, except that if the homomorphism maps any alphabetic elements to ε, the CSLs are no longer closed. Perhaps more surprising is the finding that they are all also closed under *inverse* homomorphisms, including those which thus map ε back to alphabetic elements. Since h need not be one-to-one (φ, for example, is not), h^{-1} may not be a unique function; thus inverse homomorphisms must

map strings to sets of strings, and in fact both homomorphisms and inverse homomorphisms are notationally extended to themselves apply to languages, e.g. $h(L)$. Note that, since ε is a substring of any string at any point in that string, one can use the inverse of a homomorphism mapping letters to ε as a means to insert any number of letters randomly into strings of a language, e.g. $\varphi^{-1}(001)=\{12, 13, 012, 102, 120, 0102, \cdots\}$; yet, by the closure property, languages thus enlarged (even CSLs) remain at the same level in the Chomsky hierarchy.

We can employ an even more flexible means for substituting elements in languages, based on FSAs. A *generalized sequential machine* (GSM) is an FSA whose arcs are labelled, not only with symbols from the alphabet which are expected on the input, but also with corresponding *output* symbols to which the input symbols are converted by the action of the automaton. Thus, a GSM arc might be labelled "0/1" to indicate that a 0 read on the input produces a 1 on the output. (A useful example of a GSM will be encountered in section 2.5.3.) All four Chomsky hierarchy language families and ILs as well are closed under both GSM and inverse GSM mappings, though again the CSLs are not closed for GSMs with arcs that have ε as their output.

We note in passing that 0L-systems, in keeping with their other distinctions from the Chomsky hierarchy, are closed under none of the operations described thus far. However, being ILs, we know that, for instance, the union of two 0L-languages will be an IL, and the intersection will be a CSL (excepting ε).

2.4.2 Decidability Properties. There are many questions that may be asked about languages, not all of which can be answered in the most general case by any algorithmic method—that is, there are certain *undecidable* problems related to languages. For example, we noted above that the intersection of two CFLs need not be a CFL, but of course it *may* be; it happens that determining whether it is or not for arbitrary CFLs is undecidable. It is undecidable whether one language is a subset of another, or even equal to another, for languages that are beyond regular; the same is the case for determining if two languages are pairwise disjoint (i.e. non-overlapping). Surprisingly, even the question of whether a language is empty is decidable only up through the ILs.

Perhaps the most basic question we can ask about languages is whether a given string is a member of a given language. Luckily, this question is decidable for all but the recursively enumerable languages, i.e. those specified by unrestricted grammars. This latter should not be too surprising, since in general Turing machines cannot be guaranteed to halt on arbitrary input.

Closure properties, and even more so decidability properties, suggest a motivation for studying languages in these terms, and wherever possible for using grammars to specify them that are as low on the Chomsky hierarchy as possible. Simply put, there is a marked tradeoff between the expressive

power required for languages and their general "manageability." Nowhere is this more obvious than in the task of determining membership of a given string in a given language, which, though decidable, may yet be intractable. This task of recognition is the subject of the next section.

2.5 Parsing

While automata can be used for recognition, these theoretical machines may not lead to practical implementations. The algorithmic aspect of computational linguistics is the search for efficient recognizers or *parsers* which take as input a grammar G and a string w, and return an answer as to whether w belongs to $L(G)$, the language generated by G. Many such parsers have been designed and implemented; we will mention a few of the most important.

The regular languages can be parsed in time which is $O(n)$ on the length of the input string, and in fact it is easy to see how to implement a parser based on regular expression specifications. It is also the case that the deterministic subset of CFLs can be parsed in linear time, using a class of parsers typified by the LR(k) parsers [Sudkamp, 1988]. For CFLs in general, the Cocke-Kasami-Younger (CKY) parser uses a *dynamic programming* technique to save results concerning already-parsed substrings, preventing their being reparsed multiple times. The CKY algorithm can parse any CFL in time that is $O(n^3)$ on the length of the input, though for linear CFLs it is $O(n^2)$ [Hopcroft and Ullman, 1979]. The Earley algorithm is a context-free parser with similar worst-case time complexity, but it is $O(n^2)$ for unambiguous grammars and in practice is nearly linear for many real applications [Harrison, 1978]. Modifications of the CKY and Earley parsers are often useful in proving the complexity of parsing with novel grammar formalisms.

For grammars beyond context-free, parsing is greatly complicated, and in fact we have already seen that no fully general parser is possible for unrestricted grammars, membership being undecidable. In all cases, it must be emphasized, it may be possible to write special purpose parsers that very efficiently recognize strings belonging to a specific language, even ones beyond the CFLs. The results given here are important when no restrictions are to be placed on languages, other than their membership in these broad families. This is in keeping with a philosophy that grammars should be declarative rather than "hard-wired" into an algorithm, and by the same token parsers should be general-purpose procedural recognizers. Nevertheless, some types of parsers may be better suited to a domain than others, just as backward-chaining inferencing (which corresponds to a parsing style known as *top-down*) may be better in some applications than forward-chaining (which corresponds to *bottom-up* parsing), or vice-versa.

A related field in computational linguistics, that of *grammatical inference*, attempts to develop algorithms that *induce* grammars by learning from exam-

sentence → *noun_phrase* *verb_phrase*
noun_phrase → *article* *modified_noun* | *modified_noun*
modified_noun → *adjective* *modified_noun* |
modified_noun *prepositional_phrase* | *noun*
verb_phrase → *verb_phrase* *noun_phrase* |
verb_phrase *prepositional_phrase* | *verb*
prepositional_phrase → *preposition* *noun_phrase*
noun → **man** | **boats** | **harbor**
verb → **watched** *adjective* → **old** | **kind**
article → **the** *preposition* → **in**

Figure 3. A Simple Natural Language Grammar

ple input strings, both positive and negative [Fu, 1982]. While some such approaches have been developed for RLs, no great practical success has been achieved as yet for CFLs or above, again reflecting the decreasing manageability of languages as the Chomsky hierarchy is ascended.

3 Computational Applications of Language Theory

In this section we will first briefly review the major arenas in which formal language theory has been applied computationally, and then present in more detail an application of a specific grammar formalism and parsing system to the problem of specifying and recognizing genes in DNA sequences. This will serve to motivate the remainder of our investigations.

3.1 Natural Language

Consider the natural language sentence *"The kind old man watched the boats in the harbor."* A highly simplified grammar that can specify this sentence (among many others) is given in Figure 3. Here, the top-level rule says that a sentence consists of a noun phrase followed by a verb phrase. Following this are the phrase-level rules, and finally the lexical entries—the tokens in this case being English words—given according to their parts of speech.

The study of human language has led to the creation of much more complex and specialized grammar formalisms, and parsers to deal with them. It is far beyond the scope of this work to review the rich literature that has resulted; for this the reader is referred to textbooks such as [Allen, 1987]. We will note, however, some basic results concerning the formal status of natural language. One straightforward observation is that natural language is ambiguous at many levels [Shanon, 1978], including a structural or syntactic level. For

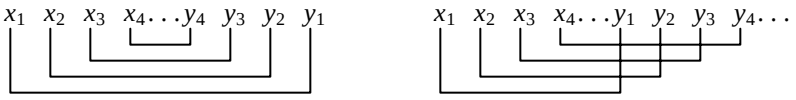

Figure 4. Dependencies

example, if I say *"I was given the paper by Watson and Crick,"* alternative valid parses could attach the prepositional phrase to the noun phrase *the paper* (e.g. to mean that someone gave me a paper written by Watson and Crick), or to the verb phrase *was given the paper* (to mean that Watson and Crick gave me some paper). Somewhat more controversial is the notion that natural language is nondeterministic, based in part on the evidence of "garden path" sentences like *"The old man the boats."* Most persons first parse *man* as a noun modified by *old*, then must backtrack upon "unexpectedly" encountering the end of the sentence, to reparse *old* as a noun and *man* as a verb. (Many, however, consider such phenomena to be jarring exceptions that prove the rule, that the human "parser" is ordinarily deterministic.)

There has been much debate on the subject of where natural language lies on the Chomsky hierarchy, but there is little doubt that it is not regular, given the apparent capacity of all human languages to form arbitrarily large sets of *nested dependencies*, as illustrated in Figure 4. An exaggerated example of such a construction would be *"The reaction the enzyme the gene the promoter controlled encoded catalyzed stopped."* Using the symbols from Figure 4, we can understand the nested relative clauses of this sentence to indicate that there is a certain promoter (x_4) that controls (y_4) some gene (x_3) that encodes (y_3) an enzyme (x_2) that catalyzes (y_2) a reaction (x_1) that has stopped (y_1). However difficult to decrypt (particularly in the absence of relative pronouns), this is a syntactically valid English sentence, and many more reasonable examples of extensive nesting can be found; these require a "stack," and thus a context-free grammar, to express. Moreover, a consensus appears to have formed that natural language is in fact greater than context-free [Schieber, 1985]; this is largely because of the existence of *crossing dependencies* in certain languages, also schematized in Figure 4, which are not suited to pushdown automata for reasons that should by now be apparent. In Dutch, for example, phrases similar to the one above have a different word order that crosses the dependencies [Bresnan et al., 1982]. Evidence that English is greater than context-free, which is generally less straightforward, is perhaps typified by the sentence (from [Postal and Langendoen, 1984]) *"Some bourbon hater lover was nominated, which bourbon hater lover*

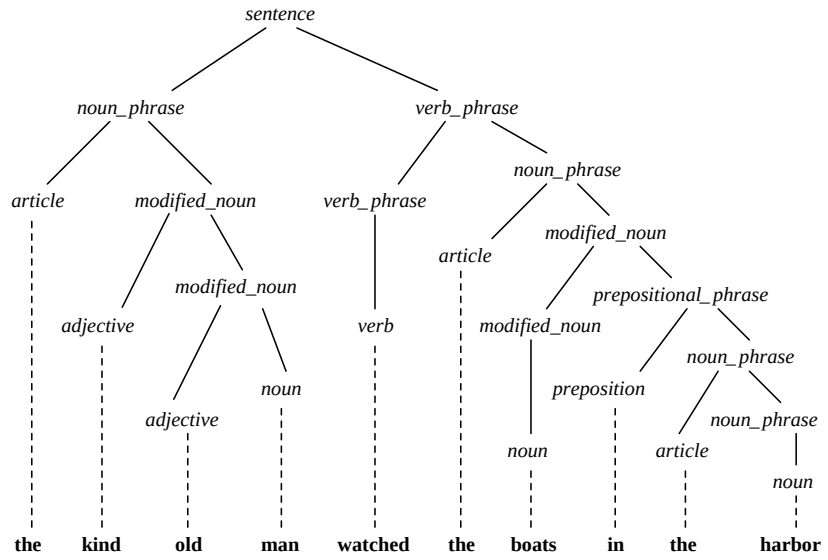

Figure 5. A Natural Language Parse Tree

fainted." Here, the instances of *hater* and *lover* form crossing dependencies, and these can theoretically be propagated indefinitely into forms such as *"bourbon hater lover lover hater . . ."* which must be duplicated in a sentence of this type, in effect forming a copy language.

3.2 Computer Languages and Pattern Recognition

Artificial languages such as computer languages are designed (whether consciously or not) to inhabit the lower reaches of the Chomsky hierarchy, for reasons of clarity and especially efficiency. The standard Backus-Naur Form (BNF) for specifying computer language syntax is, in fact, essentially a context-free grammar formalism. (A typical BNF, describing a domain-specific computer language for performing various operations on DNA, can be found in [Schroeder and Blattner, 1982].) That such languages should be unambiguous is obviously highly desirable, and they are usually deterministic CFLs as well so as to allow for fast parsing by compilers. Wherever possible, special-purpose languages such as string matchers in word processors, operating system utilities like *grep*, etc., are designed to be regular for even better performance in recognition, and overall simplicity.

Pattern recognition applications are not limited to RLs, however. The field of *syntactic pattern recognition* makes use of linguistic tools and techniques in discriminating complex patterns in signals or even images, in a manner

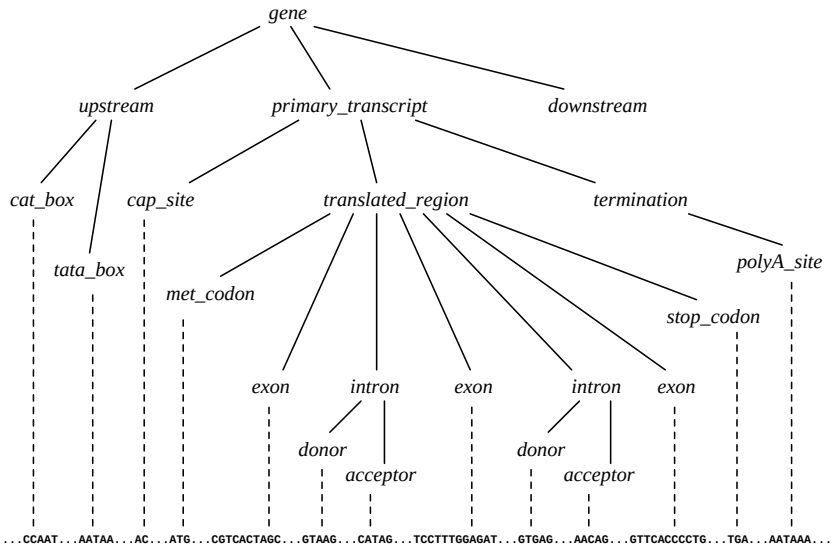

Figure 6. A Gene Parse Tree

that is more model-based and structurally oriented than traditional decision-theoretic approaches [Fu, 1982]. Specifying two-dimensional images appears to require greater than context-free power; among the formalisms used in the field for this purpose are indexed grammars [Fu, 1982; Searls and Liebowitz, 1990].

3.3 Developmental Grammars

The L-systems were in fact originally developed as a foundation for an axiomatic theory of biological development [Lindenmayer, 1968]. The process of rewriting terminals was meant to model cell division, with various elaborations on the 0L-systems allowing for developmental stage-specific variations and (in the case of context-sensitive rules) for intercellular communication. Since that time, formal linguists have explored the mathematical properties of L-systems exhaustively, while theoretical biologists have extended their application to such topics as form in plants. Quite convincing three-dimensional images of ferns, trees, and other plants, even specific species, can be generated by L-systems; these are reviewed in [Prusinkiewicz and Hanan, 1989], which also discusses the intriguing relationship of L-systems to fractals.

```
gene --> upstream, xscript, downstream.
upstream --> cat_box, 40...50, tata_box, 19...27.
xscript --> cap_site,..., xlate,..., polyA_site.

cat_box --> pyrimidine, “caat”.
tata_box --> “tata”, base, “a”.
cap_site --> “ac”.

base --> purine, pyrimidine.
purine --> “g” | “a”.  pyrimidine --> “t” | “c”.

xlate([met|RestAAs]) --> codon(met).
     rest_xlate(RestAAs), stop_codon.
rest_xlate(AAs) --> exon(AAs).
rest_xlate(AAs) --> exon(X1), intron,
     rest_xlate(Xn), {append(X1,Xn,AAs)}.

exon([]) --> [].
exon([AA|Rest]) --> codon(AA), exon(Rest).

intron --> splice.
intron, [B1] --> [B1], splice.
intron, [B1,B2] --> [B1,B2], splice.

splice --> donor, ..., acceptor.
donor --> “gt”.          acceptor --> “ag”.

stop_codon --> “tga” | “ta”, purine.
codon(met) --> “atg”.
codon(phe) --> “tt”, pyrimidine.
codon(ser) --> “tc”, base.          % etc...
```

Figure 7. A Simple Gene DCG

3.4 Gene Grammars

One of the useful byproducts of any practical parsing algorithm is a *parse tree*, illustrated for the example above in Figure 5. This is a tree-structured depiction of the expansion of the grammar rules in the course of a derivation—a *structural* representation of the syntactic information used in recognition. In practice, a parse tree or some other form of information about the parse is essential to further interpretation, e.g. for semantic analysis in the case of natural language, since otherwise a recognizer simply returns “yes” or “no.”

It is the premise of this article that DNA, being a language, should be amenable to the same sort of structural depiction and analysis; indeed, the parse tree shown in Figure 6 would appear to any biologist to be a reasonable representation of the hierarchical construction of a typical gene. This being

the case, we can fairly ask what the nature of a grammar determining such a parse tree might be, and to what extent a grammar-based approach could be usefully generalized.

To further explore this question at a pragmatic level, we have implemented such grammars using the logic-based formalism of *definite clause grammars* (DCGs). These are grammars closely associated with the Prolog programming language, and in fact are directly compiled into Prolog code which constitutes a top-down, left-to-right parser for the given grammar. The simplified gene grammar shown in Figure 7 illustrates a range of features.

The top-level rule for `gene` in this grammar is an uncluttered context-free statement at a highly abstract level. The immediately succeeding rules show how the grammar can be "broken out" into its components in a clear hierarchical fashion, with detail always presented at its appropriate level. The rules for `cat_box`, `tata_box`, and `cap_site` specify double-quoted lists of terminals (i.e., nucleotide bases), sometimes combined with nonterminal atoms like `pyrimidine`. The "gap" infix operator ('`...`') simply consumes input, either indefinitely, as in `xscript`, or within bounds, as in `upstream`.

DCGs use *difference lists*, hidden parameter pairs attached to nonterminals, to maintain the input list and to express the span of an element on it [Pereira and Warren, 1980]. For notational convenience, we will refer to *spans* on the input list using an infix operator '/' whose arguments will represent the difference lists; that is, we will write *S0/S* where *S0* is the input list at some point, and *S* is the remainder after consuming some span. We will also use an infix derivation operator '`==>`' whose left argument will be a nonterminal or sequence of nonterminals, and whose right argument will be either a list or a span to be parsed. Note that this actually represents the reflexive, transitive closure of the formal derivation operator described above. Top-level calls might appear as follows:

```
tata_box ==> "tataaa".
tata_box ==> "tatatagcg"/S.
```
(31)

Both these calls would succeed, with the latter leaving S bound to "gcg".

Features of DCGs that potentially raise them beyond context-free power include (1) *parameter-passing*, used here to build the list of amino acids in the transcript. The `exon` rule assembles sublists recursively, after which `xlate` and `xlate1` combine them to form a complete polypeptide by means of (2) *procedural attachment* in the form of a curly-bracketed call to the Prolog built-in `append`. This feature of DCGs allows arbitrary Prolog code (or other languages) to be invoked within rule bodies, extending to simple utilities, more complex search heuristics, entire expert systems, dynamic programming algorithms, or even calls to special-purpose hardware.

```
| ?- (...,gene,...):Parse ==> mushba.
[loading /sun/mn2/dbs/dna/mushba.db...]
[mushba.db loaded 0.933 sec 1,442 bases]

Parse =
...
gene:
  upstream$0:
    cat_box:282/"ccaat"
    ...
    tata_box:343/"tataa"
    ...
    cap_site:371/"ac"
    ...
  xscript:
      codon(met):
        405/"atg"
      exon:(405/501)
      intron:
        donor$2:500/"gtgaga"
        ...
        acceptor$0:606/"tctctccttctcccag"
      exon:(623/827)
      intron:
        donor$2:827/"gtatgc"
        ...
        acceptor$1:945/"cactttgtctccgcag"
      exon:(961/1087)
      stop_codon:1087/"taa"
    ...
    polyA_site$0:1163/"aataaa"
...
```

Figure 8. A Gene DCG Parse

DCGs also allow for (3) *terminals on the left-hand side* of a rule, trailing the nonterminal; they are added onto the front of the input string after such a rule parses. This feature is used by `intron` in such a way that a new codon is created when the reading frame straddles the splice site [Searls, 1988]. Rules in this form are not context-free. We can also see that procedural attachment gives the grammar Turing power, so that it can specify recursively enumerable languages, and in fact the same is true of unrestricted parameter-passing.

For large-scale search we have abandoned the built-in Prolog list structure for the input string, which is instead implemented as a global data structure in an external 'C' array. (Thus, numerical indexing replaces the DCG difference lists.) In conjunction with this, intermediate results are saved in a *well-formed substring table* (similar in principle to a CKY parser) that also prevents repeated scanning for features across large gaps. Other additions include a large number of extra hidden DCG parameters to help manage the parse, including one which builds and returns a parse tree. We have also implemented specialized operators to manage the parse at a meta level, to arbitrarily control position on the input string, and to allow for imperfect matching. In the terminal session shown in Figure 8 a search is performed on the GenBank entry "MUSHBA" containing the mouse α-globin sequence. The top level derivation operator is extended to allow calls of the form `sentence:Parse ==> input,` where the input may be specified as (among other things) a file containing sequence data, and where a parse tree may be returned via the variable `Parse`.

The grammar used was derived from that of Figure 7, but with the additional control elements described above, and much more complex rules for splice junctions that use simulated weight matrices for donors and detection of branch points and pyrimidine-rich regions for acceptors, in addition to the invariant dinucleotides. The resulting grammar, with considerable tuning, has been successful in recognizing not only mouse but human α-like globins, while ignoring pseudogenes (e.g., in the human α gene cluster "HUMHBA4"). We have also tested it against the whole 73,000+ base pair human β-globin gene region ("HUMHBB"), and were able to collect the entire cluster of five genes on a single pass that required 4.7 CPU-minutes on a Sun 3/60. A pseudogene as well as large intergenic stretches were passed over.

By "relaxing" the specifications in various ways (allowing in-frame stop codons within exons and an out-of-frame final stop codon, and loosening constraints on the splice donor weight matrix), we have also been able to study aberrant splicing that would otherwise produce untranslatable message [Searls and Noordewier, 1991]. By duplicating known β-thalassemia mutations, additional cryptic donors were recognized, most of which are observed in nature in aberrant splicing. The alternative transcription products seen experimentally were also produced by the DCG parser because of backtracking, which may also be useful for modeling the alternative transcription start sites and splicing seen in certain viruses, as well as in experiment planning applications [Searls, 1988].

The weakest links in the gene grammars developed to date are the signals for splice junctions. In a practical implementation, it may be preferable to incorporate other specialized algorithms (e.g. neural net recognizers) directly into the grammar, and procedural attachment in DCGs makes this relatively easy. The grammar still provides a very useful organizing framework, which

can serve to place such algorithms in an overall hierarchical context that captures the complex orderings and relationships among such features.

The gene grammars used for the investigations described above are written without great regard for the linguistic status of the features being parsed, and we have seen that the power of DCGs is such that the languages defined potentially may reside at any level of the Chomsky hierarchy. Nevertheless, this does not prove that the language of nucleic acids is beyond regular, and indeed most of the features specified above can be rewritten as regular expressions, however awkward they may be. The grammar form is preferable if for no other reason than that it promotes an abstracted, hierarchical view of the domain. Regular grammars have been written describing much simpler genes [Brendel and Busse, 1984], and at least one author [Shanon, 1978] has argued that the genetic language is no more than context-free, and in fact that a syntactic approach is not even necessary given its lack of structure in the usual linguistic sense. However, these arguments are based on a very limited view of biological phenomena, confined to the amino acid code itself. On the contrary, in succeeding sections it will be seen that biological sequences are rich with structural themes, both literal and linguistic.

4 Structural Linguistics of Nucleic Acids

We now proceed to consider exactly how much linguistic power is actually required to encompass various phenomena observed in nucleic acids that are literally *structural*—that is, depending on the physical nature of DNA and RNA, rather than any information encoded. The informational structure, which we will refer to as *functional* linguistics, will be discussed later. Only a minimal knowledge of the molecular biology of nucleic acids is required for this section, though a wider range of biological phenomena is cited elsewhere which is beyond the scope of this work to review; for background, readers are referred to standard textbooks such as [Watson et al., 1987; Lewin, 1987].

4.1 Properties of Reverse Complementarity

Before beginning, we will establish a notation and some basic properties of nucleic acid complementarity. We will uniformly adopt the alphabet of DNA

$$\Sigma_{\mathrm{DNA}} = \{\, \mathrm{g}, \mathrm{c}, \mathrm{a}, \mathrm{t} \,\} \tag{32}$$

and let a bar notation represent an operation corresponding to simple base complementarity, i.e. indicating bases that are able to physically and informationally *base-pair* between strands of double-helical DNA:

$$\overline{\mathrm{g}} = \mathrm{c}, \qquad \overline{\mathrm{c}} = \mathrm{g}, \qquad \overline{\mathrm{a}} = \mathrm{t}, \qquad \text{and} \qquad \overline{\mathrm{t}} = \mathrm{a} \tag{33}$$

While much of the work to follow will apply primarily to RNA structure, we will assume that features of interest are actually being examined on the DNA which encodes them. Clearly this operation can be extended over strings and constitutes a homomorphism, since we can say that

$$\overline{u} \cdot \overline{v} = \overline{(uv)} \qquad \text{for } u,v \in \Sigma^*_{\text{DNA}} \tag{34}$$

We will abbreviate (34a) as $\overline{uv}$. We can also see that this homomorphism and string reversal have the following properties:

$$\overline{(\overline{w})} = w, \qquad (w^R)^R = w, \qquad \text{and} \qquad \overline{(w^R)} = (\overline{w})^R \tag{35}$$

The composition of complementarity and reversal in (35c), which will be written as $\overline{w}^R$, is of course the "opposite strand" of a string w of DNA, since not only are the strands of a double helix complementary but they are oriented in opposite directions. Care must be taken not to treat this operation as a homomorphism, since it does not itself preserve concatenation in general:

$$\overline{u}^R \cdot \overline{v}^R \neq \overline{(uv)}^R = \overline{v}^R \cdot \overline{u}^R \qquad \text{where } |u| \neq |v| \tag{36}$$

Rather, such a string function is an *involution* [Head, 1987]. We can easily derive from the lemmas of (35) the familiar property that in essence allows nucleic acids to be replicated from opposite strands:

$$\overline{(\overline{w}^R)}^R = \overline{(\overline{w}^R)^R} = \overline{(\overline{w})} = w \tag{37}$$

We will demonstrate one other fundamental property (also noted by [Head, 1987]), concerning the special case of strings that are identical to their opposite strands, i.e. those in the language

$$L_{\text{e}} = \{\, w \in \Sigma^*_{\text{DNA}} \mid w = \overline{w}^R \,\} \tag{38}$$

We note first that any such w must be of even length, or else it would have a centermost base not identical to the centermost base of its opposite strand, since they are required to be complementary. Thus, we can divide w into two equal halves, and also conclude that

$$w = uv = \overline{w}^R = \overline{v}^R \overline{u}^R = u\overline{u}^R \qquad \text{where } |u| = |v| \tag{39}$$

(where the bar notation is now used to denote the lengths of the strings). Thus we see that L_{e} is in fact the language

$$L_{\text{e}} = \{\, u\overline{u}^R \mid w \in \Sigma^*_{\text{DNA}} \,\} \tag{40}$$

The equivalence of the languages (38) and (40) will come as no surprise to any molecular biologist, since it is simply a linguistic expression of the basic notion of *dyad symmetry.* The language L_{e} will become important in the following section.

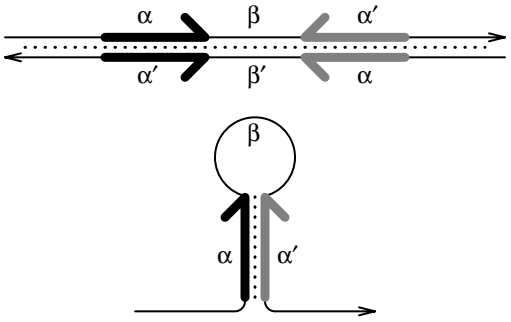

Figure 9. An Inverted Repeat

4.2 Nucleic Acids Are not Regular

Inverted repeats are prevalent features of nucleic acids, which in the case of DNA result whenever a substring on one strand is also found nearby on the opposite strand, as shown at the top of Figure 9. This implies that the substring and its reverse complement are both to be found on the same strand, which can thus fold back to base-pair with itself and form a stem-and-loop structure, as shown at the bottom.

Such base-pairing within the same strand is called *secondary structure.* It would seem that we could specify such structures with the following context-free grammar:

$$S \to bS\bar{b} \mid A \qquad A \to bA \mid \varepsilon \qquad \text{where } b \in \Sigma_{\text{DNA}} \tag{41}$$

The first rule sets up the complementary base pairings of the stem, while the second rule makes the loop. Note that disjuncts using b, here and in all subsequent grammars, are actually abbreviations that expand to four disjuncts, e.g. allowing in the first rule above every possible alphabetic substitution that maintains the required complementarity. These complementary bases establish nested dependencies between respective positions along the stem.

However, the A rule for the loop in (41) is an obstacle to further analysis, since it can specify any string and thus the resulting language is simply Σ^*_{DNA}, making it trivially regular. We will return to this issue in a moment, but in order to study the essential aspects of this language, we will first focus on the base-pairing stems and drop the A rule from (41), thusly:

$$S \to bS\bar{b} \mid \varepsilon \tag{42}$$

The resulting language may be thought of as that of idealized, gapless biological "palindromes," able to form secondary structure extending through entire strings with no loops (i.e., we imagine them having no steric hindrance

to prevent complete base-pairing to the ends of the stems, whereas in reality there is a minimal loop necessary). In fact, this is simply the language L_e of (40) representing sequences concatenated to their own reverse complements. This equivalence can be shown by simple inductions on the length of strings in (40) and the number of derivation steps used in (42); we leave this to the reader, though proofs along the same lines will be given below.

We will, however, show that L_e cannot be an RL, by proving that no FSA can recognize it. Such an FSA would, for instance, be required to accept $g^i c^i$ for all $i \geq 1$ and reject $g^j c^i$ for all $i \neq j$. Let q_n denote the node or state in which the FSA arrives after having processed a string g^n. Then we know that $q_i \neq q_j$ for all $i \neq j$, since starting from the state q_i and consuming the string c^i leads to a final node, while from q_j, consuming the same string c^i must *not* lead to a final node. Thus the FSA must have distinct states q_i and q_j for all $i \neq j$ and, since any length input is allowed, it must therefore have an infinite number of states. Since an FSA must by definition be finite, there can be no such FSA recognizing L_e, and thus L_e cannot be regular.

4.3 Non-Ideal Secondary Structure

Let us call a string *ideal* whenever, for each base type, its complement is present in the string in equal number. Languages having only ideal strings, or grammars that specify them, will also be called ideal. The grammar (42) is ideal, since any time a base is added to the terminal string, so is its complement. However, the grammar (41) is non-ideal, due to its loop rule.

In addition, (41) is inadequate as a model because in fact it accepts *any* string of any size via the loop disjunct, and can bypass the more meaningful stem disjunct entirely. One practical solution to this problem is to place constraints on the extents of these subcomponents, for instance requiring a minimum length p for the stem and a maximum length q for the loop. This reflects biological reality to the extent that inverted repeats that are too small or too far separated in a nucleic acid molecule can be expected to base-pair less readily. For a given fixed p and q, this gives rise to the language

$$L_n = \{\, uv\bar{u}^R \mid u,v \in \Sigma^*_{\mathrm{DNA}},\ |u| \geq p, \text{ and } |v| \leq q \,\} \tag{43}$$

That this is still a CFL is demonstrated by our ability to specify it as a context-free grammar, as follows:

$$\begin{aligned}
&S \rightarrow A_0 \\
&A_i \rightarrow bA_{i+1}\bar{b} && \text{for } 0 \leq i < p \\
&A_p \rightarrow bA_p\bar{b} \mid B_0 \\
&B_j \rightarrow bA_{j+1} \mid \varepsilon && \text{for } 0 \leq j < q \\
&B_q \rightarrow \varepsilon
\end{aligned} \tag{44}$$

Here, subscripted rules are meant to be expanded into multiple rules according to p and q. The A rules account for the stem, with each distinct rule "counting" the base pairs up to the minimum required, then permitting any number of additional base pairs; similarly, the B rules count the unpaired bases of the loop, but in this case impose a maximum. We will prove that this language L_{n} is not regular, by contradiction. Suppose that it were indeed regular, and let us derive a new language from it:

$$L'_{\mathrm{n}} = \phi\big(\mathbf{gc} \cup \mathbf{ggcc} \cup \mathbf{gggccc} \cup \cdots \cup \mathbf{g}^{p\text{-}1}\mathbf{c}^{p\text{-}1} \cup (L_{\mathrm{n}} \cap \mathbf{g^*a}^{q}\mathbf{c^*})\big) \quad (45)$$

where ϕ is the homomorphism based on ϕ(g)=0, ϕ(c)=1, and ϕ(a)=ϕ(t)=ε. We see that for fixed p and q each of the expressions in L'_{n} is regular, and furthermore we know that the RLs are closed under each of the operations used, i.e. intersection, union, and homomorphism. Thus L'_{n} itself must also be regular. Now let us simplify the expression (45), first examining the intersection of L_{n} with the regular expression on the right. Of all the strings generated by L_{n}, this regular expression "selects" ones that have exactly q consecutive a's, flanked by any number of g's on the left and c's on the right, and no t's at all. Since the a's thus have nothing with which to base-pair, they must all be in the loop portion, and in fact because there are q of them they must constitute the entire loop. The flanking g's and c's thus base-pair to form the stem, and being base-paired they must be present in equal numbers, greater than or equal to p. Similarly the sub-expressions on the left are a finite union of RLs containing equal numbers (less than p) of g's followed by c's. The homomorphism ϕ serves to convert g's and c's to a different alphabet and to discard the a's, leaving the language

$$L'_{\mathrm{n}} = \phi\big(\{\, \mathrm{g}^{j}\mathrm{c}^{j} \mid 1 \le j < p \,\} \cup \{\, \mathrm{g}^{i}\mathrm{a}^{q}\mathrm{c}^{i} \mid i \ge p \,\}\big) = \{\, 0^{n}1^{n} \mid n \ge 1 \,\} \quad (46)$$

But this language is the same as (7), which is known not to be regular (as can be demonstrated using essentially the same proof as in the previous section). Thus our assumption that L_{n} is regular must be false, and we may extend this result to a conjecture that secondary structure with any suitable limits placed on its non-ideal components will not be regular. (In particular, relating the non-ideal to the ideal regions, e.g. allowing them to be proportional in size, would appear to raise the resulting languages even beyond context-free.)

4.4 Nucleic Acids are neither Deterministic nor Linear

As was noted above, the nondeterministic parser inherent in DCGs is useful in dealing with empirical nondeterminism in biological systems, such as alternative splicing and other transcriptional variants. But besides this observed nondeterminism, we can now see that the structure of nucleic acids, in particular that associated with inverted repeats, is nondeterministic by its na-

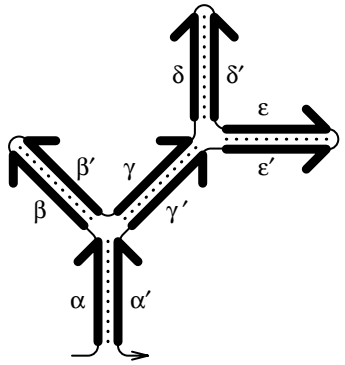

Figure 10. Recursive Secondary Structure

ture. By reasoning similar to that given above for (11), any recognizer for such structures would have to guess at the center point of the inverted repeat. The presence of a loop does not alter this result.

The grammar (42) for inverted repeats is linear; however, many more elaborate forms of secondary structure are possible, and anything with more than a single stem structure would not be linear. For example, a grammar specifying any number of consecutive inverted repeats would be simply

$$S \to AS \mid \varepsilon \qquad A \to bA\bar{b} \mid \varepsilon \tag{47}$$

Clearly this, or any other grammar specifying multiple inverted repeats, would exceed the capabilities of a one-turn PDA. Even this is not a "most general" form for ideal secondary structure, however, because it does not allow for structure within structure, which is quite common in RNA in configurations like that of Figure 10. We can propose a formal description of all such secondary structure by recursively building a set of strings of this nature.

Let us define an *orthodox* string as either ε, or a string derived from an orthodox string by inserting an adjacent complementary pair, $b\bar{b}$, at any position. The intuition behind this definition is that adding such pairs to a secondary structure will either extend the tip of a stem, or cause a new stem to "bud off" the side of a stem, and these are the only operations required to create arbitrary such secondary structure. Clearly every orthodox string is ideal. Moreover, we can specify the set of all orthodox strings, L_o, with a grammar that merely adds to (42) a disjunct that duplicates the start symbol:

$$S_o \to bS_o\bar{b} \mid S_oS_o \mid \varepsilon \tag{48}$$

That this specifies exactly the orthodox strings is shown by induction on the length of the string. The empty string ε is both orthodox and derivable

from (48). Assuming that any and all orthodox strings of length $2n$ (only even-length strings being allowed) are derivable from (48), we must show that the same is true for orthodox strings of length $2(n+1)$. For the longer string to be orthodox, it must be built on some orthodox string w of length $2n$ that, we know by the inductive hypothesis, is derivable from (48). Without loss of generality, we can assume that the derivation of w delays all ε rule applications to the end. Note also that, for every derivation step applying the first disjunct of (48) to derive the substring $b S_o \bar{b}$, we can substitute a derivation producing the substring $S_o b S_o \bar{b} S_o$ instead, since

$$S_o \Rightarrow S_o S_o \Rightarrow S_o S_o S_o \Rightarrow S_o b S_o \bar{b} S_o \Rightarrow b S_o \bar{b} S_o \Rightarrow b S_o \bar{b} \tag{49}$$

Therefore, we can ensure that in the intermediate string just before the ε rules are applied in the derivation of w, there will be S_o's flanking every terminal base, in every possible position where the next bb might be added to create the orthodox string of length $2(n+1)$. Since $b\bar{b}$ is derivable from such S_o's, this same derivation can be easily extended to produce any and all such strings, completing the inductive proof.

4.5 Nucleic Acids Are Ambiguous

We have seen that non-ideal secondary structure grammars such as (41) are ambiguous, in a way that can subvert the implicit biological meaning (since bases which rightfully could base-pair in the stem via the first disjunct can also be attributed to the loop by the second rule). We can observe a much more interesting form of ambiguity in the grammar of (48) that relates biologically to the underlying language of orthodox secondary structure, L_o. Consider the sublanguage of L_o consisting of concatenated inverted repeats:

$$L_e^2 = L_e \cdot L_e = \{ u\bar{u}^R v\bar{v}^R \mid u, v \in \Sigma_{DNA}^* \} \tag{50}$$

This in turn contains the set of ideal double inverted repeats, i.e.

$$L_d = \{ u\bar{u}^R u\bar{u}^R \mid u \in \Sigma_{DNA}^* \} \tag{51}$$

Any such string can clearly be derived from S_o as two side-by-side inverted repeats, but it follows from the equivalence of (38) and (40) that the entire string can also be parsed as a single inverted repeat, e.g. the following two leftmost derivations from the grammar (48):

$$\begin{aligned} S_o \Rightarrow S_o S_o \Rightarrow g S_o c S_o \Rightarrow ga S_o tc S_o \Rightarrow gatc S_o \\ \Rightarrow gatcg S_o c \Rightarrow gatcga S_o tc \Rightarrow gatcgatc \end{aligned} \tag{52}$$

$$S_o \Rightarrow g S_o c \Rightarrow ga S_o tc \Rightarrow gat S_o atc \Rightarrow gatc S_o gatc \Rightarrow gatcgatc$$

Note that these two derivations correspond to two alternative secondary structures available to the input string, as illustrated in Figure 11. The first derivation of (52), which spawns two S_o's, in effect describes the so-called "dumbbell" structure shown at the left, in which the two inverted repeats

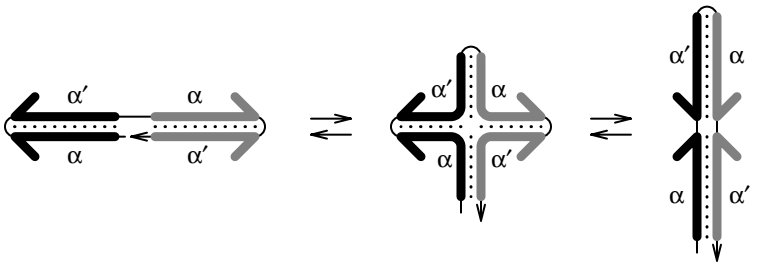

Figure 11. Dumbbell, Cloverleaf, and Hairpin Structures

base-pair separately; the second derivation, which uses a single S_o throughout, describes the uniform "hairpin" structure shown at the right. Such double inverted repeats are indeed thought to assume both structures alternatively in certain biological situations (e.g. RNAs specifically replicated by the bacteriophage T7 DNA-dependent RNA polymerase [Konarska and Sharp, 1990]), as well as intermediate "cloverleaf" structures, as shown in the center of Figure 11. In fact it can be seen that for ideal double inverted repeats of this form, a continuous series of such intermediate structures are available, base pair by base pair, between the two extremes. It is gratifying that each such secondary structure corresponds to a different partition on the set of leftmost derivations, interpreted in this manner, e.g. the following cloverleaf version of the input from (52):

$$\begin{aligned} S_o &\Rightarrow gS_oc \Rightarrow gS_oS_oc \Rightarrow gS_oS_oS_oc \Rightarrow gaS_otS_oS_oc \\ &\Rightarrow gatS_oS_oc \Rightarrow gatcS_ogS_oc \Rightarrow gatcgS_oc \Rightarrow gatcgaS_otc \Rightarrow gatcgatc \end{aligned} \tag{53}$$

This suggests a strong analogy between derivations and physical secondary structures—in fact, parse trees from these grammars can be seen as actually depicting such structure. (The extent to which alternative structures are allowed is related to the language-theoretic notion of *degree of ambiguity*.)

Of course, having found an ambiguous grammar for such features does not imply that the language containing them is *inherently* ambiguous; that would require proving that no unambiguous grammar suffices. Surprisingly, the language L_o of generalized orthodox secondary structure appears not to be inherently ambiguous, since it falls in a class of languages (the full or two-sided Dyck languages—see section 2.7.5) for which unambiguous grammars are possible [Harrison, 1978, p. 322]. However, there may well exist sublanguages of L_o which are inherently ambiguous (perhaps the language L_e^2 of (50), which is similar to the inherently ambiguous language of concatenated pairs of ordinary palindromes [Harrison, 1978, p. 240]). In any case,

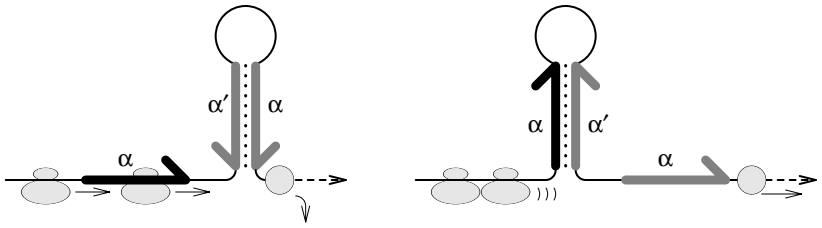

Figure 12. Attenuator Structure

we might actually prefer an ambiguous grammar that models an underlying biological ambiguity, such as alternative secondary structure, particulary when that ambiguity has functional significance.

Attenuators, for example, are bacterial regulatory elements that depend on alternative secondary structure in their corresponding mRNA to control their own expression [Searls, 1989a]; a simplified representation of their structure is shown in Figure 12. An attenuator has the capability to form alternative secondary structure in its nascent mRNA, under the influence of certain exogenous elements depicted in the figure, to establish a kind of binary switch controlling expression of a downstream gene [Lewin, 1987]. If we model an attenuator as either of two alternative languages corresponding to these states,

$$L_{\text{off}} = \left\{ uv\bar{v}^R \;\middle|\; u,v \in \Sigma^*_{\text{DNA}} \right\} \qquad L_{\text{on}} = \left\{ u\bar{u}^R v \;\middle|\; u,v \in \Sigma^*_{\text{DNA}} \right\} \tag{54}$$

then the relationship of these languages to those of (29), and of their *union* to the inherently ambiguous language of (23), is apparent. Nevertheless, this is still not a formal proof, and in fact it can be argued that L_{off} and L_{on} should actually be *intersected*, since both conditions are required to be present in the same language to produce the function described (see section 2.7.2).

Again, while we leave open the question of the formal status of nucleic acids vis-à-vis inherent ambiguity, we note that a contrived unambiguous grammar for any given secondary structure may be inferior as a model, if it fails to capture alternatives in the secondary structure. Moreover, the definitional requirement for a *leftmost* derivation may itself be irrelevant to the physics of folding, which presumably can occur simultaneously along the length of the molecule. An interesting exception to this would be the folding of nascent RNA that occurs as it is synthesized, which of course is leftmost.

Another functional theme in nature involving alternative secondary structure is *self-priming* of certain DNA molecules, such as parvoviruses [Watson et al., 1987] where the ends of double-stranded molecules are able to refold into T-shaped configurations that can "bootstrap" the synthesis of a new copy of the entire viral genome. In this case, the most fundamental process of

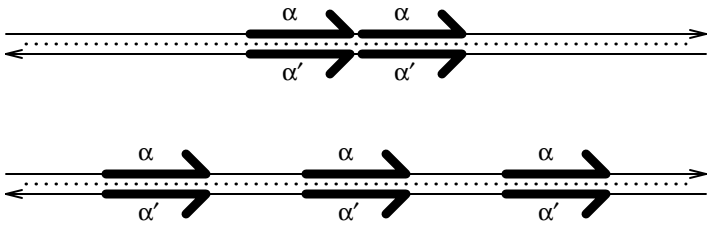

Figure 13. Tandem and Direct Repeats

replication of an organism may be viewed as depending on a kind of ambiguity in the language containing its genome (see section 2.7). We will shortly see how replication might itself result in inherent ambiguity (see section 2.5.1).

The nondeterminism of secondary structure rules out linear-time parsing, and its nonlinearity and possible inherent ambiguity would also preclude certain quadratic-time simplifications of well-known parsers. Any of the structural elements given so far could be parsed in cubic time at worst (or, indeed, recognized more efficiently by less general algorithms), but we will now offer evidence for non-context-free features that may create further complications.

4.6 Nucleic Acids Are not Context-Free

The presence (and importance) of tandem repeats and direct repeats of many varieties in DNA, as depicted in Figure 13, indicate the need to further upgrade the language of nucleic acids; these are clearly examples of copy languages, as specified in (14), which are known to require CSLs for their expression. Direct repeats entail crossing dependencies, where each dependency is in fact simply equality of the bases.

While there is thus strong empirical evidence for any general language of nucleic acids being greater than context-free, we may yet ask if there is any structural correlate, as is the case for context-free secondary structure. Several possibilities are shown in Figure 14. The illustration on the left suggests that a string of direct repeats extending infinitely in either direction could shift an arbitrary number of times, and still maintain base-paired structure with its reverse complementary string through alternative "hybridization." In practice, of course, only a few direct repeats might suffice, and in fact such misalignment in highly repetitive sequences is postulated to occur in mechanisms of change involving unequal crossing over [Lewin, 1987]. The illustration on the right of Figure 14 shows how a circular molecule could be formed by alternative base pairing between a simple tandem repeat and its

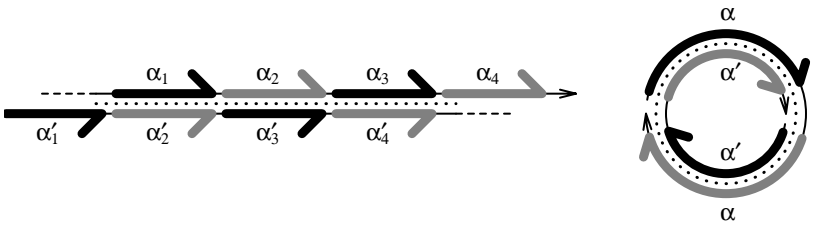

Figure 14. Structural Correlates for Direct Repeats

reverse complement. (Circular molecules, which we will have occasion to deal with again, are in fact quite important in biology; they have been suggested as a motivation to extend formal language theory to circular strings [Head, 1987].)

Are there, however, mechanisms whereby a *single* strand can form secondary structure that is not encompassed by the grammar of (48), and is thus perhaps greater than context-free? In fact, recent evidence points to "non-orthodox" forms of secondary structure, called *pseudoknots,* in many RNA species [Pleij, 1990]. Such a structure is shown in Figure 15. While each base-paired region only creates nested dependencies, the combination of the two necessitates crossing those dependencies.

To formally illustrate the consequences of this, consider an ideal pseudoknot language (i.e. one without unpaired gaps, etc.), which can be represented as follows:

$$L_{\mathrm{k}} = \left\{ uv\bar{u}^{R}\bar{v}^{R} \;\middle|\; u,v \in \Sigma^{*}_{\mathrm{DNA}} \right\} \tag{55}$$

We will prove that this language is not context-free, again by contradiction. If L_{k} were indeed a CFL, then since CFLs are closed under intersection with RLs, the language

$$L'_{\mathrm{k}} = L_{\mathrm{k}} \cap \mathbf{g}^{+}\mathbf{a}^{+}\mathbf{c}^{+}\mathbf{t}^{+} \tag{56}$$

would also be a CFL. We can see that any choice of the substring u from (55) must exactly cover the initial g's selected by the regular expression, while v must exactly cover the a's, etc. Otherwise, some substring from (55) would have to contain the boundary pairs 'ga', 'ac', and/or 'ct'; this cannot be, because each substring's reverse complement is present, and therefore so would be the pairs 'tc', 'gt', and/or 'ag', respectively, all of which are forbidden by the regular expression. We know that the length of u and thus the number of g's is equal to the length of $\bar{u}^{R}$ and the number of c's, and similarly for v and $\bar{v}^{R}$ so that in fact

$$L'_{\mathrm{k}} = \left\{ \mathrm{g}^{i}\mathrm{a}^{j}\mathrm{t}^{i}\mathrm{c}^{j} \;\middle|\; i,j \geq 1 \right\} \tag{57}$$

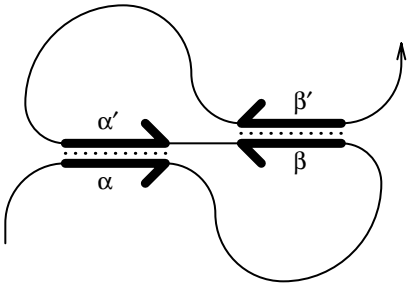

Figure 15. Pseudoknot Structure

which is related by a trivial homomorphism (under which CFLs are also closed) to the language (13b), known to be greater than context-free. Hence, L_{k} cannot be context-free.

The pseudoknot language L_{k} of (55) is clearly ideal, but cannot be orthodox because it contains strings, such as those in L'_{k}, that have no adjacent complementary bases. Thus, there exist ideal but non-orthodox secondary structure languages which are greater than context-free. We can, however, show that the most general ideal language, i.e. the set of all ideal strings (orthodox or not) L_{i}, is a CSL with ε since it is specified by the following essentially context-sensitive grammar:

$$\begin{aligned} &S_{\mathrm{i}} \rightarrow B_b\, b S_{\mathrm{i}} \mid \varepsilon \\ &B_b \rightarrow \overline{b} \qquad\qquad \text{for each } b,d \in \Sigma^*_{\mathrm{DNA}} \\ &d\, B_b \rightarrow B_b\, d \end{aligned} \tag{58}$$

This grammar can only generate ideal strings, since every b derived is accompanied by a B_b which must eventually produce exactly one $\overline{b}$. The proof that (58) generates every ideal string is by induction on the lengths of such strings. The ideal string ε derives from (58); we assume that any ideal string of length $2n$ does also, and attempt to show this for any ideal string w of length $2(n+1)$. It must be the case that $w = u\overline{b}vb$ for some $u,v \in \Sigma^*_{\mathrm{DNA}}$, and furthermore the ideal string uv of length $2n$ must derive from (58) by the inductive hypothesis. It can be seen that S_{i} can only appear once in any intermediate string of this derivation, and always at the end; thus, in a leftmost derivation the final step must be an application of the ε rule to the string uvS_{i}, in which case we can adapt this derivation to produce

$$S_{\mathrm{i}} \Rightarrow \cdots \Rightarrow uvS_{\mathrm{i}} \Rightarrow uvB_b\, bS_{\mathrm{i}} \Rightarrow uvB_b\, b \Rightarrow^{m} u\, B_b\, vb \Rightarrow u\overline{b}vb \tag{59}$$

where the penultimate derivation steps comprise sufficient applications of the final, context-sensitive rule of (58) to allow B_b to traverse v leftwards to its final position—that is, $m = |v|$. This completes the induction, as well as the

proof that the grammar of (58) generates exactly L_{i}, the set of all ideal strings.

From the results thus far concerning secondary structure, we may make the informal generalization that orthodox structure is inherently context-free, and ideal non-orthodox structure is greater than context-free. Care must be taken in extending these intuitions to specific cases, though, since subsets of languages may be either higher or lower in the Chomsky hierarchy than the original language. For example, the language generated by **(gact)*** is ideal and non-orthodox, but obviously regular, while the language of double inverted repeats, L_{d} of (51), is orthodox but not a CFL, since it also specifies direct repeats. We also note in passing, without proof, the interesting observation that for a complementary alphabet of less than four letters (e.g. if only g's and c's are used) there can be no non-orthodox ideal strings.

4.7 Nucleic Acids as Indexed Languages

The features described thus far are all encompassed by CSLs with ε, and in fact can be described by indexed grammars, which specify the IL subset of CSLs. The following indexed grammar defines the copy language of DNA (i.e., tandem repeats):

$$S \to bS^b \mid A \qquad A^b \to Ab \qquad A \to \varepsilon \tag{60}$$

(It may be noted that this simplified grammar does not strictly correspond to the formal definition of an indexed grammar, but there is an easy transformation to one that does, e.g. using stack end markers, etc.) The first rule serves to record in the indices all of the bases encountered, while the A rule "plays back" the bases in the proper order. A sample derivation from this grammar would be

$$\begin{aligned} S &\Rightarrow \mathrm{g}S^{\mathrm{g}} \Rightarrow \mathrm{gc}S^{\mathrm{cg}} \Rightarrow \mathrm{gca}S^{\mathrm{acg}} \Rightarrow \mathrm{gca}A^{\mathrm{acg}} \\ &\Rightarrow \mathrm{gca}A^{\mathrm{cg}}\mathrm{a} \Rightarrow \mathrm{gca}A^{\mathrm{g}}\mathrm{ca} \Rightarrow \mathrm{gca}A\mathrm{gca} \Rightarrow \mathrm{gcagca} \end{aligned} \tag{61}$$

Note that we can easily specify inverted repeats as well, which is not surprising since the ILs contain the CFLs. We just substitute in the grammar (60) a different rule for A:

$$S \to bS^b \mid A \qquad A^b \to \bar{b}A \qquad A \to \varepsilon \tag{62}$$

Then, following the same course as the last example derivation (61), we have

$$\begin{aligned} S &\Rightarrow \mathrm{g}S^{\mathrm{g}} \Rightarrow \mathrm{gc}S^{\mathrm{cg}} \Rightarrow \mathrm{gca}S^{\mathrm{acg}} \Rightarrow \mathrm{gca}A^{\mathrm{acg}} \\ &\Rightarrow \mathrm{gcat}A^{\mathrm{cg}} \Rightarrow \mathrm{gcatg}A^{\mathrm{g}} \Rightarrow \mathrm{gcatgc}A \Rightarrow \mathrm{gcatgc} \end{aligned} \tag{63}$$

ILs can contain an unbounded number of repeats (or inverted repeats, or combinations thereof), by simply interposing an additional recursive rule in the grammar (60). We can also specify "interleaved" repeats, as in the fol-

lowing grammar specifying the pseudoknot language L_k of (55):

$$S \to bS^b \mid A \qquad A^b \to bA\overline{b} \mid B \qquad B \to \overline{b}B \qquad B \to \varepsilon \qquad (64)$$

With this facility for handling both crossing and nested dependencies, it is tempting to speculate that the phenomena observed in biological sequences may be contained within the ILs. It has been suggested that ILs suffice for natural language [Gazdar, 1985], and it is also interesting to recall that 0L-systems, which have been so widely used to specify biological form, are contained within the ILs [Prusinkiewicz and Hanan, 1989].

5 Closure Properties for Nucleic Acids

Viewed as a kind of *abstract datatype*, nucleic acids could be usefully defined by the range of biological operations that can be performed on them. Viewed as language, it thus becomes important to understand their linguistic behavior under those operations. In this section we examine a number of known closure properties of languages under various operations that are relevant to nucleic acids, as well as some derived operations that are specific to the domain.

5.1 Closure under Replication

Consider the operation devised on strings $w \in \Sigma^*_{\text{DNA}}$ to denote the reverse complementary string, $\overline{w}^R$. Are the language families of interest closed under this operation? In other words, if we decide that some phenomenon in DNA falls within the CFLs (for example), can we be assured that the *opposite strand* will not be greater than context-free?

Recall that the bar operation is an "ε-free" homomorphism. Of the language families we have described, the RLs, CFLs, ILs, CSLs, and recursively enumerable languages are all closed under such homomorphisms; as it happens, they are also all closed under string reversal, and thus we can be confident that opposite strands will maintain the same general linguistic status. This being the case, we can design an operation on sets of strings that will *replicate* them in the sense of creating and adding to the set all their opposite strands:

$$\text{REP}(L) = \left\{ w, \overline{w}^R \mid w \in L \right\} = L \cup \overline{L}^R \qquad \text{for } L \subseteq \Sigma^*_{\text{DNA}} \qquad (65)$$

Since we have closure under union for all these language families as well, they are still closed under this replicational operation. Note that the definition of (65) accords well with the biological fact of *semi-conservative replication*, in which there is a "union" of each original string with its newly-synthesized opposite strand. Indeed, we can extend this operation to its own closure (i.e., allowing any number of applications of it), denoted as usual by an asterisk, and observe a much stronger, biologically-relevant result:

$$\text{REP}^*(L) = \text{REP}(L) \tag{66}$$

This follows from (37), and is simply a linguistic statement of the fact that, once REP has been applied to any population of strings and they are thus "double-stranded," the same strings will recur for any number of replications.

It should be noted, however, that the deterministic CFLs are not closed under either homomorphism or string reversal, so that a context-free feature that parses deterministically on one strand may be nondeterministic (though still context-free) on the opposite strand. The following suggests why:

$$L_{\text{D}} = \left\{ \text{a}\,\text{g}^i\,\text{c}^i\,\text{g}^j \;\middle|\; i,j \geq 1 \right\} \cup \left\{ \text{t}\,\text{g}^i\,\text{c}^j\,\text{g}^j \;\middle|\; i,j \geq 1 \right\} \tag{67}$$

Were it not for the initial 'a' or 't' on every string in this CFL, it would be nondeterministic for reasons described in relation to the languages of (23) and (29). However, the 'a' and 't' act as "markers" that tip off the recognizer as to what elements it should use the stack to count, making L_{D} deterministic. Note, therefore, that any homomorphism that mapped 'a' and 't' to the same element would negate the effects of the markers and leave a nondeterministic language. More to the point, string reversal moves the marker bases to the opposite ends of the strings where the recognizer will not encounter them until the end. Thus,

$$\overline{L}^R_{\text{D}} = \left\{ \text{c}^i\,\text{g}^i\,\text{c}^j\,\text{a} \;\middle|\; i,j \geq 1 \right\} \cup \left\{ \text{c}^i\,\text{g}^j\,\text{c}^j\,\text{t} \;\middle|\; i,j \geq 1 \right\} \tag{68}$$

would be recognized (in a leftmost fashion) nondeterministically. (A more formal grounding for this nonclosure proof may be found in [Harrison, 1978]). One practical consequence of this is that there may be situations where it is better to parse a string in one direction than another, particularly with a top-down backtracking parser like that of DCGs; for example, one would want to establish the presence of the invariant dinucleotides in a splice junction before searching for the much more difficult flanking signals.

Since replication as we have defined it constitutes a union of a language with its reverse complementary language, it is easy to show that unambiguous CFLs are not closed under this operation, since there may be strings in "double-stranded" sets such that we cannot know *a priori* from which strand they came. For example, the language

$$L_{\text{U}} = \left\{ \text{g}^i\,\text{c}^i\,\text{g}^j \;\middle|\; i,j \geq 1 \right\} \tag{69}$$

is a deterministic (and thus unambiguous) CFL, since a PDA could simply push the stack on the first set of g's and pop on the c's, with no guesswork required. However, when replicated this language becomes

$$\text{REP}(L_{\text{U}}) = \left\{ \text{g}^i\,\text{c}^j\,\text{g}^k \;\middle|\; i=j \text{ or } j=k \right\} \tag{70}$$

which is essentially the inherently ambiguous language of (23), necessarily having multiple leftmost derivations whenever $i=j=k$.

5.2 Closure under Recombination

Other "operations" that are performed on nucleic acid molecules include recombinatory events. For simplicity, we will confine ourselves here to primitive manipulations like scission and ligation. The latter is ostensibly straightforward, for, if we define ligation and the "closure" of ligation (i.e. the ligation of any non-zero number of strings from a language) as follows

$$\begin{aligned} \mathrm{LIG}(L) &= \{\, xy \mid x,y \in L \,\} = L \cdot L \\ \mathrm{LIG}^*(L) &= \{\, x_1 x_2 \cdots x_n \mid n \geq 1 \text{ and } x_i \in L \text{ for } 1 \leq i \leq n \,\} = L^+ \end{aligned} \tag{71}$$

then we can see that these correspond to concatenation and its positive closure over languages, and it is the case that RLs, CFLs, ILs, CSLs, and recursively enumerable languages are all closed under these operations.

It must be emphasized that this simple definition has inherent in it an important assumption regarding the modelling of biological ligation. Viewing nucleic acids as literal strings in solution, one might think that there is no *a priori* reason they should not ligate head-to-head and tail-to-tail, as well as head-to-tail as is implicit in the usual mathematical operation of concatenation. It happens, though, that these strings are not only directional, but that ligation is only chemically permitted in the head-to-tail configuration; in this instance, life mimics mathematics. As a practical matter, however, ligation generally occurs in populations of double-stranded molecules, so we must take account of the fact that in this case the strings from L in the definitions (71) will also ligate head-to-tail as reverse *complements*. Indeed we see that

$$\begin{aligned} \mathrm{LIG}(\mathrm{REP}(L)) &= \mathrm{LIG}(L \cup \overline{L}^R) \\ &= (L \cdot L) \cup (L \cdot \overline{L}^R) \cup (\overline{L}^R \cdot L) \cup (\overline{L}^R \cdot \overline{L}^R) \end{aligned} \tag{72}$$

gives all the required combinations, and uses only operations that preserve our stated closure results.

In the case of scission, we take advantage of the fact that the language families listed above, with the sole exception of the CSLs, are closed under the operations of selecting all prefixes or all suffixes of a language, i.e. under

$$\mathrm{PRE}(L) = \{\, x \mid xy \in L \,\} \qquad \mathrm{SUF}(L) = \{\, y \mid xy \in L \,\} \tag{73}$$

This being the case, we can prove closure under scission for either a single cut or for any number of cuts, by combinations of these operations:

$$\begin{aligned} \mathrm{CUT}(L) &= \{\, x,y \mid xy \in L \,\} = \mathrm{PRE}(L) \cup \mathrm{SUF}(L) \\ \mathrm{CUT}^*(L) &= \{\, u \mid xuy \in L \,\} = \mathrm{PRE}(\mathrm{SUF}(L)) \end{aligned} \tag{74}$$

The latter operation, in fact, is just the set of all substrings of L. Once again, it is interesting to note that, within the CFLs, neither deterministic nor unambiguous languages are closed under these operations, even though CFLs overall are closed.

Ligation offers one further complication, based on the fact that it may occur so as to form *circular* molecules. We will denote this variation LIG$^{\circ}$, but we are left at a loss as to how to represent it linguistically, since the strings have no beginnings. However, we *can* define the results of scission of languages formed by this operation. Assuming in the simplest case that we perform a circular ligation of each individual string in L and then cut each circle once at every possible position, we arrive at the language

$$\text{CUT}(\text{LIG}^{\circ}(L)) = \{ vu \mid uv \in L \} = \text{CYC}(L) \tag{75}$$

which is the set of circular permutations of each string. As it happens, all of the language families in the Chomsky hierarchy are closed under this operation (though, again, deterministic CFLs are not); a constructive proof of this for CFLs is given in [Hopcroft and Ullman, 1979]. Closure of LIG$^{\circ}$ really only amounts to circular ligation of repeated linear ligations, i.e. LIG$^{\circ}$(LIG*(L)), since a string can only be circularized once. Thus, our closure results still hold for this extension.

Biologists can manipulate DNA molecules by cutting them at specific sites using *restriction enzymes*, and then ligating the resulting fragments (also in a sequence-specific manner). The closure of so-called *splicing systems* under these domain-specific operations has been studied using formal language theory [Head, 1987]. Natural recombination, as between homologous chromosomes during meiosis, is an exceedingly important biological phenomenon that bears some resemblance to *shuffle* operations on languages [Hopcroft and Ullman, 1979].

5.3 Closure under Evolution

Consider the following linguistic formulations of several known modes of rearrangement at a genomic level that occur in evolution—duplication, inversion, transposition, and deletion:

$$\begin{aligned} \text{DUP}(L) &= \{ xuuy \mid xuy \in L \} \\ \text{INV}(L) &= \{ x\bar{u}^{R}y \mid xuy \in L \} \\ \text{XPOS}(L) &= \{ xvuy \mid xuvy \in L \} \\ \text{DEL}(L) &= \{ xy \mid xuy \in L \} \end{aligned} \qquad \begin{aligned} &\text{where } x,y,u,v \in \Sigma^{*}_{\text{DNA}} \\ &\text{and } L \subseteq \Sigma^{*}_{\text{DNA}} \end{aligned} \tag{76}$$

We see immediately that CFLs (and RLs, for that matter) could not be closed under DUP since this operation creates direct repeats of arbitrary length, as in (14), which are greater than context-free. What is somewhat more surprising, given the results of the previous section, is that the CFLs are also not closed under either INV or XPOS. This can be seen by the effects of the operations on inverted repeats, from which INV can make direct repeats and XPOS can make pseudoknot patterns; formal proofs of this follow.

Consider the CFL selected from among the inverted repeats—that is, from

the language L_e of (40) – by intersection with a regular expression:

$$L_{C1} = L_e \cap \mathbf{(g+c)^*at(g+c)^*} = \left\{ x\,\mathrm{a\,t}\,\overline{x}^R \;\middle|\; x \in \{\mathrm{g,c}\}^* \right\} \tag{77}$$

We can use intersection with a different RL to examine only the inversions of this language that occur over suffixes of L_{C1} (i.e. for which $y=\varepsilon$ in (76b)):

$$\mathrm{INV}(L_{C1}) \cap \mathbf{(g+c)^*at} = \left\{ xx\mathrm{a\,t} \;\middle|\; x \in \{\mathrm{g,c}\}^* \right\} \tag{78}$$

The 'at' can only arrive at the end of the string as the result of inversions of the suffix starting just before the 'at' in each string of L_{C1}. We can then use a homomorphism mapping 'a' and 't' to ε, such as ϕ given for (45), to get rid of the final at's and leave a simple copy language as in (14). Since we have arrived at a non-CFL, and every other operation used preserves CFLs, it must be the case that CFLs are not closed under inversion, and the specific case of inverted repeats yields direct repeats.

Transposition is dealt with by a similar route, first selecting a different subset of inverted repeats as our test CFL:

$$L_{C2} = L_e \cap \mathbf{g^+a^+t^+c^+} = \left\{ \mathrm{g}^i\mathrm{a}^j\mathrm{t}^j\mathrm{c}^i \;\middle|\; i,j \geq 1 \right\} \tag{79}$$

We now force transpositions that again occur over suffixes of strings in L_{C2}, such that x in (76c) covers the g's and a's, u covers the t's, v covers the c's, and $y=\varepsilon$:

$$\mathrm{XPOS}(L_{C2}) \cap \mathbf{g^*a^*c^*t^*} = \left\{ \mathrm{g}^i\mathrm{a}^j\mathrm{c}^i\mathrm{t}^j \;\middle|\; i,j \geq 1 \right\} \tag{80}$$

But this is a pseudoknot language—in fact, L'_k of (56), which we have already seen is greater than context-free. We conclude that CFLs are also not closed under transposition.

Among the evolutionary operators, CFLs are closed only under deletion. To show this, let us temporarily supplement Σ^*_{DNA} with the character §, and design a homomorphism for which $\phi(b)=b$ for $b \in \Sigma^*_{DNA}-$§, and ϕ(§)=ε. We will also set up a GSM G with transitions as given in Figure 16. Then, we see that the deletion operator can be defined as

$$\mathrm{DEL}(L) = G(\phi^{-1}(L)) \tag{81}$$

The inverse homomorphism will distribute any number of §'s in every possible position in every string of L, so we can use the first two such §'s in each resulting string as end markers for deletions, and be assured of arriving at every possible deletion, as in DEL. We accomplish those deletions with G (which also disposes of the §'s), as the reader may confirm. Since CFLs are closed under inverse homomorphism and the action of GSMs, we know that DEL(L) will be a CFL. Similar results hold for RLs, ILs, and recursively enumerable languages, though it happens that CSLs need not be closed under DEL because G is not ε-free.

Note that minor variations of G can be used to prove the closure proper-

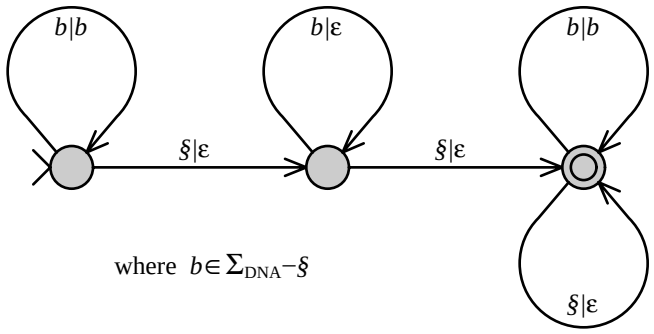

Figure 16. The Generalized Sequential Machine G

ties of the prefix and suffix operations given in (73), and in fact those results can be used to demonstrate the closure properties of the deleted fragments themselves, whether linear or circular. In addition, the definitions of *G* and ϕ may be modified to reflect other, domain-specific models in order to take advantage of the same proof methodology. For example, ϕ(§) can be defined to be a recognition sequence that delimits "directed" deletions (see section 2.8.2). Using a combination of two bracketing deletion markers, we might model the splicing that occurs in RNA processing (see section 2.3.4), or indeed even the inverse operation of inserting languages (at least RLs) into existing languages at designated points; this suggests that the evolution of interrupted genes may not in itself have contributed to their linguistic complexity.

6 Structural Grammars for Nucleic Acids

As noted, the DCG gene grammar presented previously was largely created without regard for the linguistic status of DNA, but rather as a rapidly-prototyped, reasonably efficient recognizer for "real-world" search applications. This section details our efforts to adapt logic grammars to a wider variety of biological phenomena, with formally-based conventions suitable to the domain.

6.1 Context-Free and Indexed Grammars

Base complementarity, as defined in (33), is easily implemented within DCGs by creating a special prefix tilde operator as follows:

```
~"g" --> "c".          ~"c" --> "g".
~"a" --> "t".          ~"t" --> "a".
```
(82)

Then, creating a DCG version of the formal grammar (41) specifying

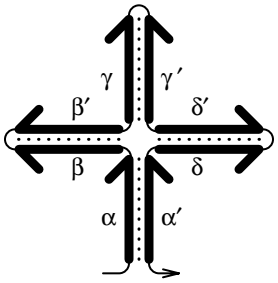

Figure 17. An n*-leaf Clover (*n*=3)*

stem-and-loop structures is straightforward:

```
inverted_repeat --> [X], inverted_repeat, ~[X]
inverted_repeat --> ... .
```
(83)

Here, the Prolog variables within square-bracketed lists indicate terminals. The gap rule represents the loop, corresponding to the rule for the nonterminal *A* in the formal grammar. We have noted that this non-ideal specification is insufficient as a model, and it is also impractical in actual parsing; however, we can implement the constrained version of (44) with little extra trouble, using parameters and embedded code to create a concise and workable (though inefficient) DCG for inverted repeats with specified minimum-length stems and maximum-length loops:

```
inverted_repeat(Stem,Loop) --> {Stem=<0},
     0...Loop.
inverted_repeat(Stem,Loop) --> {Next is Stem-1},
     [X], inverted_repeat(Next,Loop), ~[X].
```
(84)

It is just as easy to transcribe other formal grammars, e.g. that of (48) representing generalized orthodox secondary structure, to their DCG equivalents. Again, a practical implementation of the DCG form would allow us to add length constraints, gaps, and other conditions to take account of "real-world" factors. We can also write grammars for more distinctive (that is, less general) features, such as structures in the nature of "*n*-leaf clovers" like the one illustrated in Figure 17:

```
cloverleaf --> [X], cloverleaf, ~[X] | leaves.
leaves --> leaf, leaves | [].
leaf --> [Y], leaf, ~[Y] | [].
```
(85)

As was noted above, indexed grammars can be thought of as context-free grammars that are extended by the addition of a stack feature to nonterminal

elements; thus they are easily implemented in DCGs, by just attaching parameters in the form of Prolog lists to nonterminals. A DCG-based indexed grammar corresponding to (60) would be

```
tandem_repeat(Stack) --> [X],
     tandem_repeat([X|Stack]).
tandem_repeat(Stack) --> repeat(Stack).
repeat([]) --> [].
repeat([H|T]) --> repeat(T), [H].
```
(86)

while, to make the `repeat` rule instead play back the reverse complement of the sequence stored on the stack, we could substitute the rule corresponding to (62) as follows:

```
complement([]) --> [].
complement([H|T]) --> ~[H], complement(T).
```
(87)

Calling the top-level rule with an empty stack gives the desired results. An indexed grammar expressing the *n*-leaf clover of (85) would be

```
cloverleaf(Stack) --> [X], cloverleaf([X|Stack]).
cloverleaf(Stack) --> leaves([]), complement(Stack).
leaves([]) --> [].
leaves(Stack) --> [X], leaves([X|Stack]).
leaves(Stack) --> complement(Stack), leaves([]).
```
(88)

Compared with the context-free DCG of (85), this notation becomes somewhat clumsy, a problem we will address in the next section.

6.2 String Variable Grammars

We have developed a domain-specific formalism called *string variable grammar* (SVG) which appears to handle secondary structure phenomena with significantly greater perspicuity than indexed grammars [Searls, 1989a]. SVGs allow *string variables* on the right-hand sides of otherwise context-free rules, which stand for substrings of unbounded length. An example of an SVG implemented within an extended DCG formalism would be:

```
tandem_repeat --> X, X.
```
(89)

This requires only some minor modification to the DCG translator to recognize such variables as what amounts to indexed grammar nonterminals, with the Prolog variable itself representing the nested stack [Searls, 1989a]. The variables, on their first occurrence, are bound nondeterministically to arbitrary substrings, after which they require the identical substring on the input whenever they recur. We can also generalize our rules for single base

complements, to recognize the reverse complement of an arbitrary string. This allows rules of the form

```
inverted_repeat --> X, _, ~X.
```
(90)

Here we have used an anonymous string variable to denote the gap, since it is the case that `... --> _ .` Now, the rules for direct and inverted repeats—features that intuitively share a similar status in this domain—can also assume a very similar grammatical structure.

Returning to our example of the *n*-leaf clover of the grammars (85) and (88), we can now write a much more concise grammar in the form of an SVG:

```
cloverleaf --> X, leaves, ~X.
leaves --> [] | Y, ~Y, leaves.
```
(91)

We also offer economical SVG representations of the attenuator structure of Figure 12 and the pseudoknot structure of Figure 15:

```
attenuator --> A, _, ~A, _, A.
pseudoknot --> A, _, B, _, ~A, _, ~B.
```
(92)

The use of string variables can be augmented in various ways. For instance, by allowing them to be passed as parameters, we can specify an unbounded number of direct repeats:

```
direct_repeats(X) --> X, _, direct_repeats(X).
direct_repeats(X) --> X.
```
(93)

Then, by defining compositions of string variables (e.g. `~(~X) --> X.`), we can do such things as specify any number of strictly alternating inverted repeats:

```
inverted_repeats(X) --> X, _, inverted_repeats(~X).
inverted_repeats(X) -->  [].
```
(94)

We have recently shown that SVGs used in the manner described up to this point specify languages that are formally contained within the ILs, contain the CFLs, and furthermore can be parsed in $O(n^3)$ time using a variation on the Earley parser [Searls, manuscript in preparation].

6.3 Structural Grammar Examples

The SVG formalism makes it possible to describe and recognize much more complex patterns of secondary structure, such as the following specification of the 3' (right-hand) end of tobacco mosaic virus RNA, covering 177 nucleotides of which two thirds are base paired [Pleij, 1990]:

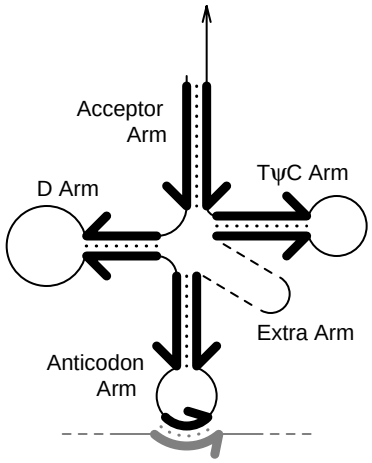

Figure 18. tRNA Secondary Structure

```
tmv_3prime --> A, _, B, ~A, _, ~B,
     C, _, D, ~C, _, ~D, E, _, F, ~E, _, ~F, _,
     G, _, H, I, _, J, _, ~J, ~I, ~G, _, ~H,
     K, _, ~K, L, _, M, ~L, _, ~M, _.
```
(95)

This pattern consists of three consecutive, contiguous pseudoknots (corresponding to the variable sets *A/B, C/D,* and *E/F),* another pseudoknot *(G/H)* whose gap contains an inverted repeat with a bulge *(I/J),* followed by another inverted repeat *(K)* and a final pseudoknot *(L/M).* Another example, adapted from [Gautheret et al., 1990], is the following grammar describing a consensus secondary structure for a class of autocatalytic introns:

```
group_I_intron --> _, A, _, B, ~A, _, C, _,
     E, _, F, _, ~F, _, ~E, G, _, ~G, _,
     D, _, ~C, _, H, _, ~H, _, ~D, _,
     I, _, ~I, _, ~B, _.
```
(96)

This structure contains two pseudoknot patterns; one *(A/B)* spans the entire sequence and in fact brackets the cleavage sites *in vivo*, while the other *(C/D)* is embedded in a series of stem-and-loop structures and variants.

These rules could obviously be written more hierarchically, using the appropriate rules for "phrases" (such as `inverted_repeat` from (90), `pseudoknot` from (92b), and others), but even as they stand they are significantly more readable than other grammar formalisms would be. Nevertheless, they do lack the necessary length constraints to make them practi-

```
tRNA(AA) --> Stem@7, "t", base, d_arm, base,
     anticodon_arm(AA), extra_arm, t_psi_arm,
     ~Stem$1, acceptor_arm.

d_arm --> Stem@4, "a", purine, 1...3, "gg",
     1...3, "a", 0...1, ~Stem$1.

anticodon_arm(AA) --> Stem@5, pyrimidine, "t",
     anticodon(AA), purine, base, ~Stem$1.

extra_arm --> 3...20, pyrimidine.

t_psi_c_arm --> Stem@4, "gttc", purine, "a",
     base, pyrimidine, "c", ~Stem$1.

acceptor_arm --> base, "cca".

anticodon(AA) --> ~[X,Y,Z],
     {codon(AA)==>[X,Y,Z], ! ; AA=suppressor}.
```

Figure 19. A tRNA SVG

cal. In order to demonstrate a real parsing application, we will use a slightly simpler case, that of transfer RNA, which is illustrated in Figure 18 and whose typical cloverleaf structure is represented abstractly by the following SVG:

```
tRNA(Codon) --> AcceptorArm, _, DArm, _, ~DArm, _,
     AnticodonArm, _, ~Codon, _, ~AnticodonArm,
     _, TpsiCArm, _, ~TpsiCArm, ~AcceptorArm, _.
```
(97)

Here, a parameter is used to return the codon identity of the tRNA, which is the reverse complement of the *anticodon* by which it recognizes the triplet on the mRNA specifying a particular amino acid.

The *E. coli* tRNA SVG listed in Figure 19 is a more practical version [Searls and Liebowitz, 1990], again using string variables for the secondary structure, but now combined with grammar features specifying known conserved bases or base classes. Despite these lexical constraints, most of the information available has to do with the folded structure of the tRNA, which causes nested dependencies to be evolutionarily conserved even where primary sequence is not. We have here used an infix control operator '@' to specify the length of the *Stem* string variables. The reverse complementary stretch uses the '$' operator to constrain the *cost* of the match, and here indicates that up to one mismatch is allowed in the stem. The grammar now returns a parameter AA indicating the actual amino acid the tRNA will carry;

```
| ?-test_tRNA(Codon,Start,End).
Parsing Arg-tRNA-1 (76bp)...    Parsing NKV region (730bp)...
Parse succeeded in 16 ms:       Parse succeeded in 583 ms:
Codon = arg,                    Codon = lys,
Start = 1,                      Start = 272,
End = 76 ;                      End = 347 ;

Parsing Asn-tRNA-1 (76bp)...    Parse succeeded in 434 ms:
Parse succeeded in 16 ms:       Codon = val,
Codon = arg,                    Start = 480,
Start = 1,                      End = 555 ;
End = 76 ;
                                Parse succeeded in 434 ms:
                                Codon = suppressor,
       ...                      Start = 558,
                                End = 633
```

Figure 20. Parses of tRNAs (left) and Genomic DNA (right)

this is determined in the `anticodon` rule using *recursive derivation* [Searls, 1989a], i.e. by parsing the triplet inside curly braces in the body of the rule. This will fail on a stop codon, as occurs in bacterial *suppressor* mutations, in which case the latter fact is returned.

This grammar was actually created from the ground up in a few hours, using 17 known bacterial tRNA sequences in isolation as a "training set." Starting with an overly constrained model based on the idealized textbook representation, the bounded gaps and cost parameters were adjusted until the entire set parsed. This is shown through the use of a higher-level Prolog rule which retrieves successive database entries, measures their length, and applies the derivation operator, keeping track of CPU time, as shown at the left in Figure 20. The grammar was then tested on genomic sequences containing tRNA gene clusters, as shown at the right of Figure 20. In this and one other gene region, all seven known genes parsed on the first try. In each case the codon was identified correctly, including a suppressor mutation [Searls and Liebowitz, 1990].

An approach similar to this one is currently being pursued in the laboratory of Dr. Ross Overbeek at Argonne National Laboratory [R. Taylor, personal communication], in the domain of ribosomal RNA molecules. These are much larger and more complex than tRNA, with dozens of stem-and-loop structures and several pseudoknots. The specialized parser being developed there will assist in the classification of new molecules, using a grammar derived from an alignment that takes account of "covariances" or dependencies preserved over evolution. We are currently investigating parsing strategies

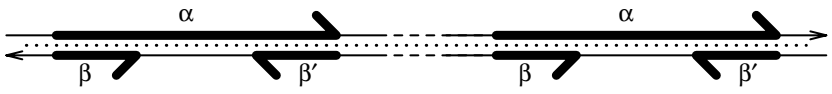

Figure 21. A Transposable Element

that would make generalized SVG parsing practical [Cheever et al., 1991].

6.4 Superpositional Grammars

Transposable elements such as *copia* typically have long terminal repeats that have "superimposed" on them terminal inverted repeats, as illustrated in Figure 21. An SVG that specifies this case would be

```
transposon --> X, Y, ~X, _, X, Y, ~X.          (98)
```

However, a better representation is as follows:

```
transposon --> W, _, W, {(X, _, ~X) ==> W}.    (99)
```

This description uses the recursive derivation to "subordinate" the inverted repeats to the more dominant terminal direct repeats—a better reflection of the semantics of the domain, since the direct repeats are typically much larger and better matches, and in fact the inverted repeats are not even always present. Other SVGs given previously can be similarly restated using recursive derivation to suggest different interpretations. For example, pseudoknots may occur in a form illustrated in Figure 22, where there is a *coaxial stacking* of the two base-pairing regions to form a quasi-continuous double helix [Pleij, 1990]. The following rule for this form of pseudoknot, it may be argued, tends to emphasize the continuity of the central stretch, and its relationship to the flanking complementary regions:

```
pseudoknot --> A, _, ~AB, _, B, {(A, B) ==> AB}.  (100)
```

We have previously presented similar sorts of "reinterpretations" of attenuator structure [Searls, 1989a], which are better at capturing the dual nature of these sequences, in that they use a recursive derivation to specify the alternative secondary structure separately.

Recursive derivation, used in this way, allows substrings to be parsed more than once within a single overall parse. We can generalize this to a notion of *superposition* of grammar elements, by defining an appropriate operator '&' (after [Pereira and Shieber, 1987]) as follows:

```
X & Y --> W, {X ==> W}, {Y ==> W}.             (101)
```

This superposition operator, which requires its operands to exactly coincide on the input, allows for some novel views on structures discussed before:

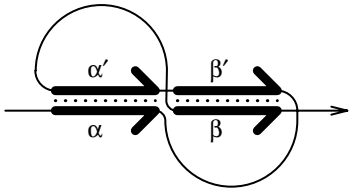

Figure 22. Coaxial Stacking in a Pseudoknot

```
tandem_inverted_repeat --> W & ~W.
double_inverted_repeat --> (X, ~X) & (Y, Y).
```
(102)

The first rule expresses the fact that the superposition of a sequence with its own reverse complement is an ideal inverted repeat, per the language L_e of (38). The second rule shows how an ideal double inverted repeat, as in L_d of (51), may be specified as the superposition of an inverted repeat with a direct repeat.

Superposition in effect acts to "disconnect" elements of the grammar from the usual strict consecutivity, as do gaps. In combination, these two features permit, for instance, specifying a promoter that has a stretch of Z-DNA (a change in the twist of the double helix that can occur where there are alternating purines and pyrimidines) occurring anywhere within it, even in superposition to other important lexical elements – which in fact is likely to be the case:

```
promoter & (_, zDNA, _)
```
(103)

Thus, superposition may prove to be an important element of functional grammars, which we will examine in the next section. For example, Z-DNA is actually most often associated with enhancers, which are even more "loosely connected" in that they can occur anywhere in the general vicinity of gene promoters, in either orientation, sometimes as direct repeats. Promoters themselves, in fact, can overlap the transcription units they control (cf. RNA polymerase III promoters), and even coding regions can coincide in certain phage [Lewin, 1987]. This suggests a need for a general capability to specify arbitrary relations between the spans of different features, similar to Allen's interval calculus [Allen, 1983]. In fact, the superposition and gap operators suffice. We can, for instance, combine them to create a subsumption ordering of alternative interpretations for an element *X* "preceding" an element *Y:*

```
(X,Y)   ≤   (X,_,Y)   ≤   ((X,_)&(_,Y))
```
(104)

The first case, in which *Y* begins immediately after *X* ends, is subsumed

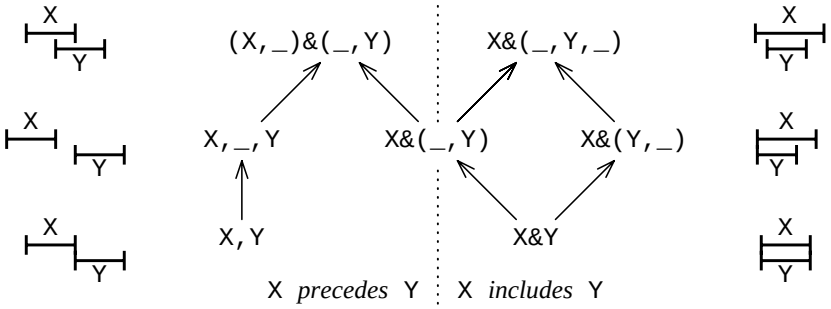

Figure 23. Partial Order of Superposition

by the second case, where *Y* can begin any time after *X* ends; this in turn is subsumed by the third case, which only requires that *Y* not begin before *X* does, nor end before *X* does. We have generalized this to a partial order, illustrated in Figure 23, that is arguably complete with respect to possible relations between spans of features [Searls, 1989b].

If, in the definition (101), *X* and *Y* were not string variables but were instantiated to the start symbols of two distinct grammars (which is allowed by the definition) then clearly *X&Y* would produce the *intersection* of the languages defined by those grammars [Pereira and Shieber, 1987]. The consequences of this will be explored in the next section.

7 Functional Linguistics of Biological Sequences

To this point we have dealt formally only with the *structural* nature of nucleic acids, which is amenable to linguistic formulation because of its relative simplicity; we will find that a *functional* or informational view of the language of biological sequences is less clear cut. This in no way weakens the results presented to this point. The closure properties derived for operations on nucleic acids, for example, apply to *any* language encoded in DNA or in any other string for which those operations are defined. Rather, the greater richness of the language of genes and proteins indicates all the more the need for a well-founded descriptive paradigm. Moreover, it will be seen that the most interesting aspects of biological languages may reside at the point where structural and functional components interact.

A functional view will also allow us to expand our horizons beyond the relatively local phenomena of secondary structure, to large regions of the genome or even entire genomes (represented formally, perhaps, as strings derived by concatenation of chromosomes). This will allow us in turn to reason linguistically about processes of evolution, at least at a conceptual level.

It may be supposed that this distinction between structural and functional linguistics corresponds to the conventional one drawn between syntax and semantics. There is much to recommend this, insofar as gene products (i.e. proteins) and their biological activities may be thought of as the *meaning* of the information in genes, and perhaps entire organisms as the meaning of genomes. On the other hand, the gene grammars presented earlier clearly demonstrate a syntactic nature, and as such grammars are further elaborated with function-specific "motifs" it may be difficult to make a sharp delineation between syntax and semantics. Ultimately, the semantics of DNA may be based on evolutionary selection; a certain view of syntax may allow sequences that do not support life (or not very well), just as syntactically-valid English sentences may nevertheless be nonsensical. The discussion that follows will not attempt to resolve where such a line should be drawn, though the potential utility of the distinction should perhaps be borne in mind.

7.1 The Role of Language Theory

In examining the functional aspects of the language of biological sequences, it becomes important to set out more precisely the goals of a language-theoretic approach. There are at least four broad roles for the tools and techniques of linguistics in this domain: *specification, recognition, theory formation,* and *abstraction.* By specification we mean the use of formalisms such as grammars to indicate in a mathematically and computationally precise way the nature and relative locations of features in a sequence. Such a specification may be partial, only serving to constrain the possibilities with features that are important to one aspect of the system. For example, published diagrams of genes typically only point out landmarks such as signal sequences, direct and inverted repeats, coding regions, and perhaps important restriction sites, all of which together clearly do not completely define any gene. However, a formal basis for such descriptions could serve to establish a *lingua franca* for interchange of information, and a similar approach may even extend to description of sequence analysis algorithms, as will be seen in a later section.

Moreover, such high-level descriptions can merge into the second role for linguistics, that of recognition. This simply refers to the use of grammars as input to parsers which are then used for pattern-matching search—that is, syntactic pattern recognition—of what may be otherwise uncharacterized genomic sequence data. We have seen that, in practice, these uses of linguistic tools tend to depart from the purely formal, for reasons of efficiency, yet a continued cognizance of the language-theoretic foundations can be important. As an example, the discovery of pseudoknots in RNA has spurred the development of new secondary structure prediction algorithms to improve on programs that, possibly without the developers explicitly realizing it, were

limited to dealing with context-free structures [Abrahams et al., 1990].

The third role of linguistics is for the elaboration of *domain theories* that in some sense model biological structures and processes. In this case, other grammatical objects in addition to terminal strings (e.g. nonterminals, productions, and even parse trees) would have specific biological semantics attributed to them, and would be developed as postulates that are testable by parsing actual positive and negative exemplars. A number of possibilities along these lines will be discussed in section 2.9 of this article. Machine learning techniques could be expected to be most useful with regards to this role, to the extent that they could be made to assist in theory formation and modification.

The fourth role of linguistics, that of abstraction, can be seen as the most conceptual insofar as its goal would be an understanding of the language of biological sequences viewed mathematically, purely as sets of strings that are at some level meaningful in a biological system. One way to define such a language would be to imagine the set of all genomes that exist in viable organisms; however, this is severely limiting insofar as there are probably many such strings that have never existed, yet would support life. This distinction, between the set of strings that exist and the set that *can* exist, parallels one drawn in natural language, between *performance* and *competence* [Chomsky, 1965]; performance refers to actual instances of the use of language, while competence refers to the intrinsic capabilities of users to generate and recognize a language. An orientation toward the latter, in any domain, can be expected to lead to more universal, intensional descriptions than an approach based simply on inventories of extant strings. Of course, such incomplete sets of instances may be important sources of information in developing linguistic descriptions, e.g. consensus sequences for regulatory regions. In many cases, though, we may derive notions of competence by observing the biological machinery that manages the strings, e.g. transcription, translation, etc. As long as our knowledge of these phenomena remains incomplete, however, these languages must remain abstractions, particularly at the level of genomes. Still, we will see that they may constitute tools for abstract reasoning and thought experiments, and the sensation that they are unfathomable must not discourage the practical application of linguistic techniques, and the ideas gleaned from this type of analysis, in the other roles described above.

7.2 The Language of the Gene

In some situations genes are superimposed so as to create ambiguity, e.g. in the cases of multiple start sites for transcription, alternative splicing, and even "nested" genes. Thus, over the same stretch of DNA there would be multiple leftmost derivations for any grammar specifying a gene, with each derivation corresponding to a gene product. Such ambiguity suggests that the corresponding language is nondeterministic, and thus not regular. However, it must be emphasized that this is not in itself a formal proof that the

language of DNA is not regular, since we have prejudiced our grammar by requiring that it capture the notion of "gene" as a nonterminal, indeed one that corresponds to a single gene product. As was the case for the formal language of (19), there may be other, perhaps less intuitive grammars that are unambiguous, specifying the same language as the ambiguous gene-oriented grammar. For example, much of the observed variation is post-transcriptional in nature, so that it may be that the ambiguity is not actually inherent at the DNA level, but resides in other cellular processes, beyond the gene itself. Thus, perhaps a transcript-oriented grammar might be unambiguous. However, we know that transcription itself can vary over time within the same region, as in the case of overlapping "early" versus "late" transcripts in certain viruses; even at the level of coding regions there is overlap, including instances of multiple reading frames. Thus, there seems to be good empirical evidence that any grammar related to genes, or purporting to model underlying biological processes at the gene level, would not be regular.

We arrive at this notion of ambiguity of gene products by viewing derivation as analogous to gene expression. In terms of information encoded, however, we must ask if such superposition is *required* of the language, rather than simply allowed. The importance of this is that it would necessitate the *intersection* of languages, under which some important language families are not closed. Returning briefly to a structural theme, consider attenuators (Figure 12), which we gave as examples of strings that are ambiguous as regards the language of secondary structure since they allow alternative folding. However, from a functional perspective, the genes that use attenuators require this secondary structure for the regulatory mechanism to work, so that the language must in fact intersect the two cases. Since the mechanism depends on orthodox secondary structure, it is CFLs, as in (54), that are being intersected, but the resulting language is greater than context-free because it necessarily contains direct repeats. While it happens to be an IL, it is a fact that any recursively enumerable language can be expressed as a homomorphism of the intersection of two context-free languages, so that there are potentially even more serious linguistic consequences to superposition of non-regular elements.

The process of gene expression by its nature suggests that genes are superpositional in another sense, reflecting the successive steps of transcription, processing, and translation, all encoded within the same region. To the extent that we wish any descriptive grammars to model these underlying processes, which occur at different times, in different places in the cell, and using different mechanisms, it would seem that such processes should be represented by separate, well-factored grammars. The projection of all the corresponding functional and control elements for these processes onto the same region of DNA should then be captured in a requirement that the respective grammars all parse that DNA successfully. Note that this would

also make it much easier to model situations such as the transcriptional variants that produce untranslatable messages, as described for the globin gene grammars above; all that would be required would be to alter the RNA processing grammar (that recognizes splice junctions) and drop the translation grammar altogether.

If we accept that separate cellular processes should be modelled by distinct grammars, and that the underlying language represents the superposition of the resulting distinct languages, then again the language may tend upwards on the Chomsky hierarchy by virtue of intersection. If it is CFLs that are being intersected, it may also be the case that we can never know the final status of the language, since it is undecidable whether the intersection of CFLs is a CFL or not. For that matter, even if we could arrive at a context-free grammar that completely described all aspects of a gene, we might not be able to show that it was non-regular, since in general it is undecidable if nondeterministic CFLs (or above) are equivalent to some RL [Hopcroft and Ullman, 1979].

7.3 The Language of the Genome

The notion of derivations corresponding to gene products would appear to be a useful one, since it formally establishes the analogies between parsing and gene expression, and between parse trees and gene structure, which are inherent in the first sample gene grammars given above. It also allows us to adapt the discussion to wider questions of the differential control of gene expression in different tissues and developmental stages. For example, if we equate successful parsing with gene expression, we must concede that a substring that is a gene at one time may not be a gene at another. This is troublesome, unless we view genes in the context of the genome as a whole. If the genome is inherently ambiguous, then multiple global derivations could correspond to particular cell types at particular times and under particular conditions. Any given derivation may or may not call for the gene subderivation in question. From this viewpoint, it might be better to name the corresponding nonterminal *expressed-gene* rather than simply *gene*.

Does this mean, though, that in any *given* fixed global state of differentiation, etc., genes and gene expression may yet be deterministic? For, at a local level the apparent ambiguity of overlapping genes, or of expressed vs. unexpressed genes, does not mean that such an ambiguity necessarily exists at any given time in the cell; there may be external, regulatory factors that "tip off" some cellular recognizer and thus specify one or the other of the available parses. In this model there could be distant elements specifying such regulatory factors in an overall genomic language, acting against ambiguity that may otherwise be present within isolated segments. Indeed, grammar-based approaches have been proposed for simulating gene regulatory systems [Searls, 1988], and for modelling their genomic arrangement using

transformational grammars (see section 2.9) [Collado-Vides, 1989b]. However, such mechanisms by and large exert their effects via exogenous elements (such as DNA-binding proteins) whose biochemical activity would seem to be necessarily "ambiguous," if only at threshold levels. It is difficult to imagine a language recognizer sophisticated enough to precisely simulate regulation that ultimately depends on the physical chemistry of molecules moving through a cell. Thus, whatever the cell might do to chart its fate deterministically, would seem to be inaccessible to any linguistic *description* of the genome out of context.

Perhaps the most telling evidence for this view is the totipotency of germline DNA, and the pluripotency of many somatic cells. That is, not only must the genome be ambiguous because it has the capacity to specify a wide variety of cell types at different times and places in development, but any counterargument based on a notion of deterministically programmed differentiation fails in the face of the many examples of dedifferentiation and developmental plasticity in biology—as clear a case for nondeterminism as could be wished. Therefore, we are left with a strong sense that the language of genes and even of the genome as a whole, must be at least context-free.

Recently an interesting proof has been offered for gene regulatory systems being greater than context-free, based on the fact that there need be no particular spatial relationship on the genome between genes coding for soluble regulatory elements and the genes those elements regulate [Collado-Vides, 1991b]. This being the case, an array of such regulatory genes and their target genes, which clearly form dependencies, are presumably free to arrange themselves so as to cross those dependencies, so that the language describing such arrangements could not be a CFL. (This is formally argued using a method of proof involving the *pumping lemma* for CFLs, to be described in section 2.8.1.)

7.4 The Language of Proteins

Proteins also have three-dimensional structure, whose nature suggests that the functional language of proteins may in fact be structural in the same sense as nucleic acids, with similar linguistic consequences. Figure 24 depicts a hypothetical folded protein molecule, illustrating in a highly schematic way the conformational relationships among major secondary structure features like α-helices (the cylinder at the bottom), β-strands (the arrows in the center), and β-turns (the "kink" iconified at the upper right).

Pattern-directed inference systems like Ariadne [Lathrop, Webster and Smith, 1987] have been used to detect amino acid sequences that are likely to produce such structures, combining statistical evidence for the features themselves with a hierarchical model of their higher-level ordering, captured in what amounts to a regular expression. Such an approach must necessarily deal with patterns seen on the unfolded string of amino acids, but clearly

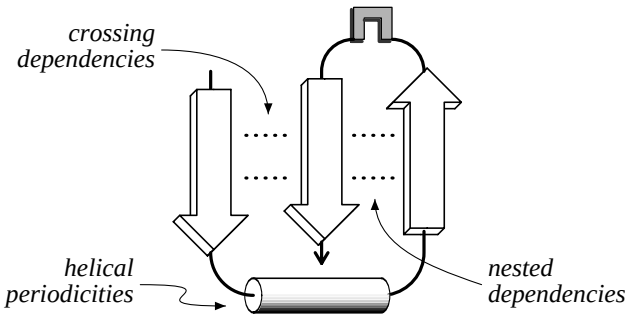

Figure 24. Protein Structure

these physical features also interact with each other in their three-dimensional conformation, not only through hydrogen bonding but also where charged moieties are brought into juxtaposition, or space-filling interactions occur, etc. These may be expected to display correlated changes over the course of evolution—i.e., dependencies.

Such interactions are suggested by the dotted lines between β-strands in Figure 24; note, however, that if those dotted lines are extended as the molecule is unfolded into a linear representation, the interactions on the right exhibit nested dependencies, and those on the left crossing dependencies, as in Figure 4. As we have seen, the former are characteristic of CFLs, and the latter of CSLs. Other such dependencies may be found, for instance, in α-helices which have one face in a hydrophobic milieu, and the other in a hydrophilic one; this will result in a series of periodic crossing dependencies.

The nested and crossing dependencies we have illustrated in inverted and direct repeats in nucleic acids are much more straightforward, corresponding to complementarity and equality of individual bases, respectively. Nevertheless, the dependencies in proteins, though more complex and varied (e.g. charge, bulk, hydrophilicity, catalytic activity within active sites, etc.) are likely to be tremendously important in terms of structure and function. Thus, it would appear that more sophisticated linguistically-based approaches to protein structure would be well-advised.

How might such non-regular functional languages interact with the non-regular secondary structures that occur in nucleic acids? They may, of course, be completely segregated, with the former confined to coding regions and the latter to control regions, introns, and structural species like tRNA and rRNA which have no polypeptide products. It is interesting to speculate, however, that nature with its usual parsimony may have elected to overlap phenomena on the DNA, for instance by favoring processed mRNA species that form secondary structure, for purposes of stability, transport, etc. We

know that the resulting language intersection may act to increase the linguistic complexity of the system, particularly since both contributing languages may have nested dependencies, which in the case of simple inverted repeats do lead to a promotion from CFLs to ILs. In the next section, however, we will explore the intriguing possibility that functional CFLs may not even require context-free grammars, beyond what already exists in nucleic acid secondary structure, for their expression.

7.5 Structurally-Derived Functional Languages

The grammar of (48), which describes ideal orthodox secondary structure in nucleic acids, defines one *particular* CFL. It is interesting to note, though, that the capability to form secondary structure, as embodied in this grammar, can be "harnessed" to express other CFLs. We have seen this in several proofs, which have used intersection with RLs together with homomorphisms to arrive at distinct CFLs, such as (45) which produces (7). As another example, consider languages consisting of true (as opposed to biological) palindromes, (11). Using the language L_{O} of orthodox secondary structure determined by the grammar (48), and a homomorphism based on $\phi(g)=\phi(c)=0$, and $\phi(a)=\phi(t)=1$, we can see that

$$L_{\mathrm{P1}} = \left\{ ww^R \,\middle|\, w \in \{0,1\}^* \right\} = \phi\left(L_{\mathrm{O}} \cap \mathbf{(g+a)^*(c+t)^*}\right) \tag{105}$$

Again, we have generated a new CFL from the generic language of secondary structure, and it would appear that this might be a fairly general capacity. However, for this example we have been forced to use all four bases—purines for the front of the palindrome, and pyrimidines for the back—raising the question of whether the utility of this tactic will be limited by the size of the DNA alphabet. For instance, it might appear that we would be unable to use L_{O} to express the language of true palindromes over an alphabet of size four:

$$L_{\mathrm{P2}} = \left\{ ww^R \,\middle|\, w \in \{0,1,2,3\}^* \right\} \tag{106}$$

As it happens, though, we can *encode* this larger alphabet into dinucleotides via a homomorphism ψ, defined as follows:

$$\begin{array}{llll} \psi(0)=\mathrm{gg} & \psi(1)=\mathrm{ga} & \psi(2)=\mathrm{ag} & \psi(3)=\mathrm{aa} \\ \psi(\hat{0})=\mathrm{cc} & \psi(\hat{1})=\mathrm{tc} & \psi(\hat{2})=\mathrm{ct} & \psi(\hat{3})=\mathrm{tt} \end{array} \tag{107}$$

As before, we will use purines for the front of the palindrome and pyrimidines for the back, but this time we use a dinucleotide to encode each digit in the final language. We must distinguish front and back digits for the moment, using the hat notation, in order for ψ to be a function, but we can strip off the hats later with another homomorphism:

$$\phi(x) = \phi(\hat{x}) = x \quad \text{for } x \in \{0,1,2,3\} \tag{108}$$

In order to make use of the encoding, we must in fact apply ψ as an *inverse* homomorphism (under which CFLs are also closed). With these exertions, we see that it is indeed possible to specify the desired language:

$$L_{P2} = \phi\psi^{-1}\left(L_o \cap \mathbf{((g^+a)(g^+a))^*((c^+t)(c^+t))^*}\right) \tag{109}$$

In fact, with appropriate encodings we should be able to specify any desired final alphabet, and with more sophisticated regular expressions we could take advantage of the ability of L_o to specify iterated or nested structures as well. The remarkable thing about these specifications is that a variety of CFLs are being expressed using the "general purpose" stack mechanism of secondary structure together with only an RL in the primary sequence and some "interpretation" $\phi\psi^{-1}$.

If the notion of the interpretation as a composition of a homomorphism with an inverse homomorphism seems odd, note that nature already uses encodings of this type, in the form of amino acid codons:

$$\begin{aligned} &\psi(\text{ser}_1)=\text{ucg} \quad \psi(\text{ser}_2)=\text{uca} \quad \psi(\text{ser}_3)=\text{ucc} \quad \psi(\text{ser}_4)=\text{ucu} \\ &\psi(\text{phe}_1)=\text{uuc} \quad \psi(\text{phe}_2)=\text{uuu} \quad \psi(\text{met}_1)=\text{aug} \quad \text{etc.}\ldots. \\ &\qquad \phi(x_i)=x \quad \text{for } x\in\{\text{ser},\text{phe},\text{met},\cdots\} \end{aligned} \tag{110}$$

where ψ now ranges over triplets from the slightly different alphabet of RNA, $\Sigma_{RNA}=\{g,c,a,u\}$. In this case, the interpretation $\phi\psi^{-1}(w)$ for $w\in\Sigma^{3n}_{RNA}$) will yield the corresponding polypeptide of length n (ignoring questions of alternative reading frame and stop codons). Here ψ establishes the encoding, and ϕ captures the degeneracy of the triplet code. It is easy to imagine other homomorphic interpretations of a similar nature being embodied, for instance, in DNA binding proteins involved in gene regulation (which in fact are often associated with regions of dyad symmetry).

This leads us to the question of whether *any* CFL, e.g. arbitrary functional languages, could be expressed by an RL superimposed on sequence with secondary structure, together with some interpretation to act as an "adaptor" to the new domain. An important characterization theorem [Chomsky and Schutzenberger, 1963] states that any CFL can in fact be specified as a homomorphism of the intersection of some RL with a language belonging to a family known as the *semi-Dyck* languages. A semi-Dyck language D_r consists of all well-balanced, properly nested strings of r types of parentheses; for example, for D_3 consisting of ordinary parentheses, square brackets, and curly braces, we would have

$$\begin{aligned} [\{\}([])]\{[]\} &\in D_3 \\ (\}\{)][&\notin D_3 \\ \{[(\})] &\notin D_3 \end{aligned} \tag{111}$$

The grammar describing semi-Dyck languages is tantalizingly close to

that of (48) for L_o, since open parentheses must be matched by closed ones in a manner quite similar to base-pairing in secondary structure. Moreover, the minimum alphabet required for a semi-Dyck language to express any CFL, in order to be able to encode larger alphabets via inverse homomorphisms [Harrison, 1978], is one with two types of parentheses for a total of four elements—exactly what nucleic acids provide. However, nucleic acid secondary structure in fact represents a full or two-sided Dyck language, i.e. one for which corresponding parentheses facing *opposite* directions, as in (111b), can also pair. In addition, we know that non-orthodox secondary structure is allowed, such as pseudoknots, which are analogous to (111c). Thus, we must leave open the question as to what the exact expressive power of this paradigm may be, not to mention the question of whether any use is made of it *in vivo*.

8 Evolutionary Linguistics

Evolution is a process that provides many interesting complications in the linguistic analysis of biological systems, as suggested by the closure results observed for the evolutionary operators of (76). In this section we will investigate some of those complications, show how grammars may be applied to describe not only evolution itself but also algorithmic tools used to compare strings that have undergone evolutionary change, and finally, discuss the prospects of extending phylogenetic analysis from strings to languages.

8.1 Repetition and Infinite Languages

We have seen from the closure results given previously that typical evolutionary rearrangements may in the right circumstances lead to a "jump" up the Chomsky hierarchy. For example, duplications create copy languages, which are not CFLs. However, we must take care to note that, simply because a language contains strings with duplications, does not mean that it is greater than context-free—once again, unbounded duplications must be *required* (or, in an evolutionary sense, actively maintained) for this to be so.

In fact, it can be shown that, even in an RL, sufficiently long strings *must* contain substrings that are allowed to occur there as tandem repeats and still leave the resulting string within the given RL. To wit, for the FSA recognizing an RL to recognize any string longer than the number of nodes or states in that FSA, some of those nodes will of necessity be visited more than once, so that there must be a *cycle* in the directed graph of the FSA. This being the case, it must also be possible to traverse that cycle *any* number of times, and thus the original string can have any number of tandem repeats at that position, and still be guaranteed to be in the RL specified by that FSA. This reasoning, a variation on the "pigeonhole principle," is known as the *pumping lemma* for RLs. There is a similar result for the CFLs, commonly

used to prove non-context-freeness, which essentially says that for sufficiently long strings derived from a context-free grammar some nonterminal must recur in the same subderivation, and this subderivation can thus be "pumped" any number of times.

This in itself need not have far-reaching consequences, since the repeat may only be of length one—in fact, simple gaps in the DCGs we have given satisfy the requirement. However, it does raise an issue related to the arbitrary extent of the repeats. If an RL contains only strings shorter than the number of nodes in its FSA, it need not have a cycle, nor a tandem repeat. This would necessarily be a finite language, and in fact *any* finite language is an RL; this can be seen from the fact that a finite set of strings can be specified by a finite, regular grammar by simply listing every terminal string in the language as a disjunct arising from the start symbol.

Thus, if the language of DNA is indeed not regular, it must be infinite. This is an assumption that has been implicit in the grammars we have written to this point, which perhaps should be examined. It could be argued that the set of all DNA molecules (or genes, or genomes, etc.) that have ever existed is finite, so that the abstraction of the language of DNA is regular. However, recall that our preferred notion of language as abstraction deals with the *capacity* of such languages to encompass all syntactically correct variations. DNA must be potentially non-regular, certainly to the extent one believes that it can specify an infinite variety of life—i.e. that there can be no complete list of possible genomes, such that no additional genome is imaginable that is different by even a single nucleotide from one already in the list. The fact of evolution adds particular force to this argument, when we realize that it is possible for the language to evolve entirely new capacities; indeed, it has apparently done this over time, e.g. at the point that eukaryotes arose.

It might also be argued that a biological language must constrain the lengths of strings, since there are practical problems with arbitrarily large genomes. For example, the bacteriophage lambda protein capsid places a limit on the size of the genome it must encapsulate; not much more than 50,000 nucleotides can physically fit inside. (In fact, a form of pumping lemma applies to this phage: since the total protein in the capsid exceeds the coding capacity of the genome, it follows that capsid proteins must be "repetitive", i.e. many identical proteins are employed in a crystalline array to achieve a large enough structure to contain the genome [Watson et al., 1987].) Thus, there would seem to be a finite number of possible phage lambda genomes. However, one can imagine mutations that alter the capsid proteins to allow an additional nucleotide insertion or two, and it becomes difficult to say where the limits are, in a system which contains the potential for respecifying the rules of its own game. Again, if we widen this to include every species that could theoretically exist, a finite language is even harder to conceive, though there may be no formal nonexistence proof. In

broader terms, even the alphabet is conceivably subject to change, as indeed may have occurred in prebiotic evolution.

Such discussions may only be of theoretical interest, since if any succinct specification is indeed possible for the potential combinatoric variation of even the relatively minuscule phage lambda genome, it may well require a powerful grammar that is "artificially" regularized with constraints, but which in outward form and every other respect is non-regular. The invocation of semantic constraint or selection might well serve this purpose. Retreating somewhat from this broadest possible abstraction, though, we can examine the case for infinite individual features, such as inverted repeats. There are probably only finitely many Type II restriction enzyme recognition sites, for example, since these are mostly inverted repeats either four or six nucleotides in length, which is about the limit of what these proteins can span on the DNA (ignoring for the moment the possibility of "changing the rules"). Are other inverted repeats of interest also limited in similar ways, e.g. is there a longest practicable stem structure in an RNA molecule? Even if a consensus for such an absolute maximum could be arrived at, it would seem that this misses the point that the self-embedding rule permitting unlimited recursion expresses the mechanical *capacity* of nucleic acids to form these structures in arbitrary lengths, and to properly capture the nested dependencies they entail.

We began this section by saying that the fact that strings in a language contain duplications does not imply that that language is not a CFL or RL. On the other hand, we can infer from the pumping lemmas that any CFL or infinite RL must allow strings with arbitrary numbers of duplications. It is only if direct repeats of arbitrary extent are for some reason required by the genome that it must be greater than context-free on this account. One sense in which duplications may be said to be required is an evolutionary one, since a primary mechanism of adaptation and change is for a gene to be duplicated and then for the copies to diverge. In fact, we can view the strings that are "pumped" in the pumping lemma as genes themselves, specified at a general enough level to allow divergence after they are duplicated. (They need not be completely general, though—a specific globin gene, for instance, can be thought of as having been pumped in this manner to create the globin gene regions.) For that matter, a diploid genome itself may be said to be a copy language, with the duplication required for the generation of diversity by recombination between homologous chromosomes.

The argument that duplications are required since they reflect a mechanism of evolution is somewhat indirect, if not circular; we could make a stronger case that the functional language of DNA is greater than context-free if functional duplications were required by the physiology of the cell. One example might be immunoglobulin variable region gene copies; though they are not exact duplicates, they serve the same function and their arrangement is required to generate diversity economically. Gene *amplification* is

another mechanism whereby gene copy numbers (in this case, exact copies) increase or decrease according to physiological demand. Once again, we see that the *generation* of specific such duplications can occur by pumping RLs or CFLs, but any *requirement* that duplications of arbitrary composition be a feature of a general functional language in order for organisms to survive would seem to raise the abstracted language beyond context-free.

8.2 Mutation and Rearrangement

A simple point mutation can be modelled grammatically by a rule that produces a side effect on the input string, e.g. through the use of terminal replacement in DCGs:

```
point_mutation([From],[To]), [To] --> [From].
```
(112)

This rule consumes no net input, but effectively just overwrites the base *From* with *To*. Such rules can be used in actual derivations by leaving uninstantiated the remainder portion of the difference list or span, e.g.

```
(_, point_mutation(X,Y), _) ==> Input/Output.
```
(113)

where *Input* but not *Output* is initially bound, will produce every version of *Input* in which *X*'s have been mutated to *Y*'s. We can write other grammars to delete and insert bases:

```
deletion(X) --> [X].
insertion(X), [X] --> [].
```
(114)

We can also generalize these rules for single base mutations to use string variables instead for "block" mutations, e.g.

```
substitution(From,To), To --> From.
```
(115)

With this expanded repertoire, we can succinctly represent a range of genomic rearrangements that occur on an evolutionary scale, corresponding to the formal definitions of (76):

```
duplication, X, X --> X.
inversion, ~X --> X.
transposition, Y, X --> X, Y.
deletion --> X.
```
(116)

This then allows us to write the most "top-level" rule of all, that for evolution itself:

```
evolution --> [] | event, evolution.
event, X --> X,
(inversion | deletion | transposition | duplication).
```
(117)

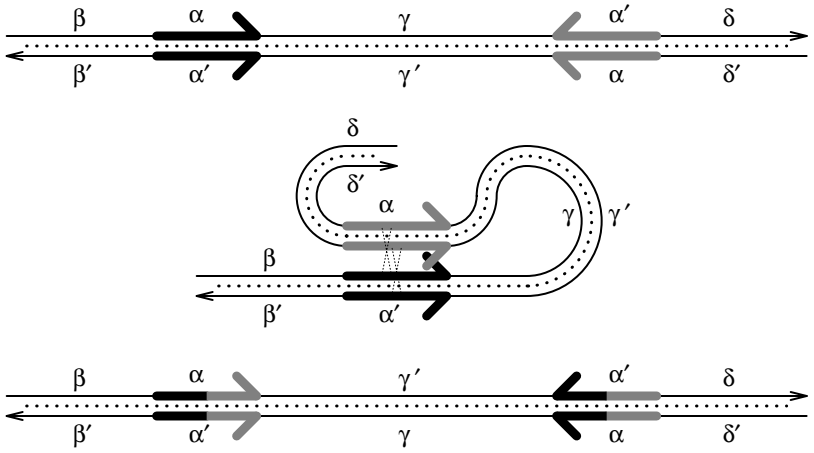

Figure 25. Inversion

This simply states that *evolution* consists of an *event* followed by more evolution. The rule for *event* is just a disjunctive list of possible rearrangements, and the variable *X* allows for arbitrary excursions down the molecule to the site where the event is to occur. We have employed a version of this grammar, which uses a random number generator for event and site selection, to simulate such evolution at the level of blocks of sequence.

More sophisticated rearrangements can also be described. For instance, an arbitrary number of duplications and reduplications can be accomplished with

```
duplication, X, X --> X | duplication, X.        (118)
```

which consumes a string and replaces two copies, but can also recursively call itself first. Combined with other forms of mutation, such a rule could, for instance, model saltatory replication involving duplications of duplications, etc., postulated to occur in mouse satellite DNA evolution [Lewin, 1987].

Some inversions are thought to occur as a result of homologous recombination between inverted repeats, as illustrated in Figure 25; examples include the tail protein of phage *Mu*, and the flagellar antigen of *Salmonella* [Watson et al., 1987]. This situation can be described using what amounts to a literally context-sensitive rule (though it is in fact unrestricted in format):

```
inversion, R, ~I, ~R --> R, I, ~R.               (119)
```

Similarly, regions between direct repeats, such as transposable elements, may be excised as circular elements, as shown in Figure 26. Excision can be

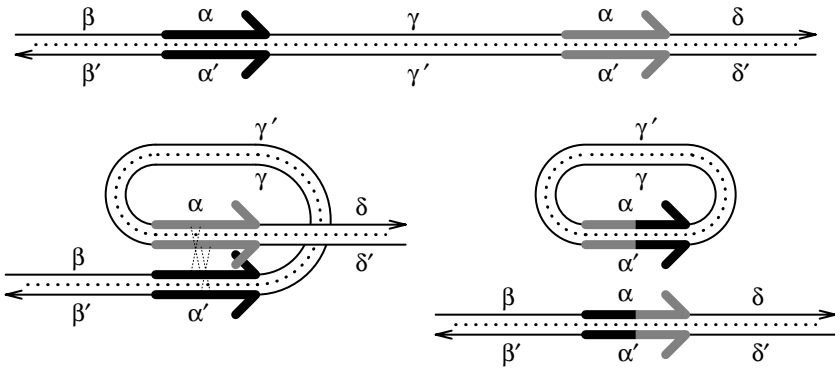

Figure 26. Excision and Integration

specified using string variable replacement, returning the excised circular fragment as a parameter, *O*:

```
excision(O), S --> S, X, S,
     {(S,X)==>O/O}.
```
(120)

We concatenate *S* and *X* using recursive derivation *generatively*, i.e. running the parse backwards. What is more, the structure that is generated, `O/O`, is a Prolog list that has itself as its own remainder, that is, a circular list. While these are awkward to handle in practice, they allow us to write a rule for the reverse reaction, that of a site-specific *integration* of a circular molecule at some substring `S` which it has in common with a linear molecule:

```
integration(O), S, X, S --> S,
     {(_,S,X,S,_)==>O/O}.
```
(121)

These and other grammars describing genome-level rearrangements are described in greater detail in [Searls, 1989a]. Our experience indicates that seemingly arbitrary such rearrangements can be quite concisely specified using SVGs. Though in a pure logic implementation they are not practical for large scale parsing, the grammar framework should be a good one in which to house more efficient lower-level algorithms to make this practicable, in a manner that will be described below.

Many of the rules in this section are unrestricted in format, though the string variables complicate the analysis somewhat. However, we can see that in one sense any grammar describing evolutionary change must in fact be greater than context-sensitive, by examining the phenomenon of recursive duplication as suggested by the grammar of (118). The ability to create any number of duplications of arbitrary substrings on the input indicates that no

linear-bounded automaton could recognize such a language, since these automata are limited to space which is linear in the size of the input, while the grammars can "grow" a given input string to an arbitrary extent. This does not mean that the strings of a language which is the *product* of an evolutionary grammar are necessarily greater than context-sensitive, since we have seen that we can recognize any number of direct repeats with an SVG that falls within the ILs.

8.3 Comparison of Strings

The use of grammars for sophisticated pattern-matching search was proposed above, where it was also demonstrated that they are likely to be better-suited to the domain, at least in terms of expressive power, than current regular-expression based systems. Another form of search that is even more prevalent in molecular biology, however, is based on the detection of similarities between strings, rather than between a pattern and a string.

In the example grammars given above, it was seen that a simple cost function could be used to allow some degree of mismatching in the course of a parse. Generalizing this to allow not only base substitutions but insertions and deletions (collectively, *indels*) we can conceive of a derivation from string to string, e.g.

```
"gaataattcggctta"$Cost ==> "gacttattcgttagaa"   (122)
```

where the "cost" would be, for instance, the *string edit distance* between the strings, or the total number of substitutions and indels required in the derivation; note how this linguistic formulation is different than the input/output model of mutation described above. We can implement this as a DCG:

```
[]$0 --> [].
[H|T]$Cost --> [H], T$Cost.    % bases match; zero cost
[H|T]$Cost --> [X], {X\==H}, T$Sub, {Cost is Sub+1}.
[_|T]$Cost --> T$Ins, {Cost is Ins+1}.          (123)
String$Cost --> [_], String$Del, {Cost is Del+1}.
```

However, there will be a very large number of such parses, of which we are only interested in the minimum cost parse. The corresponding minimal parse tree will indicate a probable *alignment* between the initial and terminal string, which may be significant should they be evolutionarily related. We can use the Prolog "bagof" operation to collect all parses and determine the minimum:

```
best(From$Cost ==> To) :-
     bagof(Cost,(From$Cost ==> To),Bag),        (124)
     minimum(Bag,Cost).
```

```
Str1$Cost --> input(Str0),     % consult chart first
     {chart(Str0,Str1,Cost-Move)}, !,
     path(Move,Str1).
_$0 --> [].                    % end of input string
[]$0 --> _.                    % end of test string
[X|Y]$Cost --> input([H|T]),   % recursively find best path
     { Y$Sub0 ==> T, (X==Y -> Sub=Sub0; Sub is Sub0+1),
       [X|Y]$Ins0 ==> T, Ins is Ins0+1,
       Y$Del0 ==> [H|T], Del is Del0+1,
       minimum([Sub-sub,Ins-ins,Del-del],Cost-Move),
       assert(chart([H|T],[X|Y],Cost-Move)), ! },
     path(Move,[X|Y]).

input(S,S,S).   % extracts input list from difference lists

path(sub,[_|R]) --> [_], R$_.  % performs specified types
path(ins,X) --> [_], X$_.      % of moves on input string
path(del,[_|R]) --> R$_.       % relative to initial string
```

Figure 27. A Dynamic Programming Alignment Grammar

but this is of exponential complexity, due to the large amount of wasteful backtracking and reparsing entailed. We have investigated the characteristics of CKY-based algorithms for finding the minimum-cost parse, and find that this can be accomplished in $O(n^3)$ time, with the performance improving the better the fit of the original strings [Searls, unpublished results]. Others have described "error-correcting" parsers based on Earley's algorithm that will find minimum cost parses for arbitrary grammars (not just the string derivations above), also in $O(n^3)$ time [Aho and Peterson, 1972].

These results compare unfavorably with dynamic programming algorithms for minimum-distance alignment that are currently used in molecular biology, which execute in $O(n^2)$ time, and can be further improved by techniques involving preprocessing, hashing, suffix trees, etc. However, the parsing algorithms offer the opportunity for generalization to pattern-matching search at a higher level of abstraction than terminal strings, for instance permitting differential weighting of features, mutations involving entire features, and so on. Moreover, we have also been able to implement several traditional "algorithmic" approaches in grammar form, such as the dynamic programming alignment algorithm given in Figure 27.

This implementation of the simplest form of distance-optimal alignment algorithm [Sellers, 1974] uses the Prolog database to record current best

scores and "moves" at each position of a comparison matrix between two strings, and prevents reparsing the same path multiple times. This grammar is clearly not optimal, and in fact it is not likely that any list-structured language could compete with procedural languages, but the ease and flexibility of the approach suggest the utility of grammars for rapid prototyping and modification of such algorithms. We have done this with a number of algorithms not ordinarily thought of as amenable to parsing, such as Fast Fourier Transforms [Searls, 1989b]. When embedded in higher-level grammars, the algorithms can then be tuned at leisure and eventually replaced with procedural code or hardware, while retaining the linguistic framework where hierarchical abstraction is of greatest benefit. As noted, we are exploring such an approach that would use signal processing hardware to implement primitive operations on string variables very efficiently [Cheever et al., 1991].

8.4 Phylogeny of Languages

Once we have embraced the notion of languages described abstractly rather than as collections of instances, we can perhaps begin to extend to the former more of the analytical tools already applied to the latter. One such tool would be phylogenetic trees, which are currently developed in general for single genes treated as strings. As illustrated at the left in Figure 28, representative strings from different species may be compared to find all the pairwise evolutionary distances, and then a tree created which postulates ancestral sequences and connections among them in such a way as to, for example, minimize the total change required. Exactly how such trees should be constructed, and distances between strings measured, is controversial and an area of active research, but it would seem that any effective notion of distance between two objects ought to conform to the basic mathematical axioms of a *metric space*; given some function δ which measures the distance between *a* and *b*, this would require that:

$$\begin{aligned} &\delta(a,b) = \delta(b,a) && \textit{symmetry} \\ &\delta(a,b) = 0 \ \textit{iff} \ a=b && \textit{identity} \\ &\delta(a,b) \leq \delta(a,c) + \delta(c,b) && \textit{triangle inequality} \end{aligned} \tag{125}$$

In fact, many (but not all) common methods of measuring string edit distances based on simple mutational models do adhere to these axioms.

Consider the possibility of a phylogenetic tree of languages which, instead of measuring degree of mutational change over individual strings, somehow measured distances between abstracted descriptions, e.g. grammars. Thus, it might be possible to focus the concept of evolutionary distance at a higher level, for instance describing major rearrangements as in the evolutionary grammars above, but perhaps also dealing with structural and functional aspects of the differences between organisms and groups of organisms. This

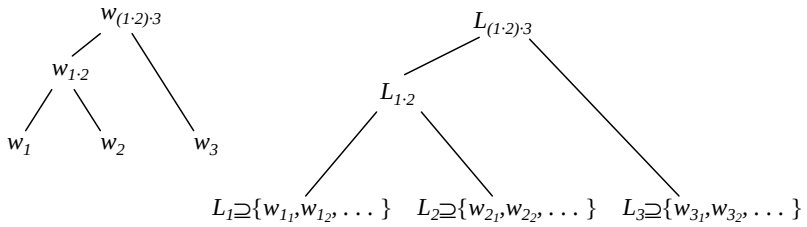

Figure 28. Phylogenetic Trees of Strings (left) and Languages (right)

idea is illustrated at the right in Figure 28, where the leaves of the tree represent sets of strings rather than individual strings. While one could in theory simply collect all the genomes from each of a number of individuals in a species and call this the language, we can by now see the virtues of concise, possibly grammar-based descriptions that freely allow variation within a species and constrain only what is critical to the "definition" of that species. Putting aside for the moment the known difficulties of such grammatical inference, we consider some of the formal consequences of such an idea.

We see first that, while the languages we are creating for each species will have many characteristics in common, they should nevertheless be pairwise disjoint; any language that claims to describe horses should contain no instance of a human being, and vice versa. Moreover, the non-leaf languages describing postulated ancestors should also in general be disjoint from present-day species. That is, we must avoid the mistake of inferring ancestors that simply *subsume* all their descendants, or else we will not have captured descriptions of change. Note, however, that we may choose to compare languages other than those of species. By generalizing languages to higher taxonomic levels, e.g. for eukaryotes versus prokaryotes, we would be distinguishing much more fundamental cellular machinery than we would by generalizing only to the level of humans versus chimpanzees, or even vertebrates versus invertebrates.

Finally, we would need a distance metric δ acting over languages, or grammars specifying those languages. With this, however, we can see some potential difficulties should the languages in question be non-regular. For we know that, given arbitrary CFLs or CSLs L_1 and L_2, it is in general undecidable whether $L_1 = L_2$. How, then, can we establish a δ which we are assured does not violate the identity axiom of (123b) whenever $\delta(L_1, L_2) = 0$? Thus, languages may not be so easy to compare. In fact, it is also undecidable whether L_1 and L_2 are pairwise disjoint, so we may not even be able to tell if our languages or their ancestors are truly distinct.

Thus, while the ideal of comparing abstract descriptions of genes or genomes holds great appeal, there are serious practical and theoretical problems to be overcome. Nevertheless, there are many avenues to explore, such as restricting the comparison to regular aspects of these languages, perhaps by confining the area of interest to specific phenomena. This approach has been used by [Overton and Pastor, 1991; Pastor, Koile and Overton, 1991], who restrict their attention to instantiated parse trees describing major features of genes and their regulatory regions in predicting the locations of those features in novel genes. Another approach is to focus on simple lexical and prelexical elements, following the methodology of the field of classical linguistics where prototypic languages are inferred from changes in basic vocabulary sets. Similar techniques have been used for biological sequences, for instance, by [Brendel et al., 1986; Pietrokovski et al., 1990]. As for the methodology of inducing grammars, a recent proposal would use logic grammar "domain descriptions" of DNA regulatory regions as a starting point for connectionist learning programs, which would in effect "tune" the general grammar by modifying and refining it [Noordewier and Shavlik, personal communication]. Others have used model-based learning to derive grammar-like descriptions of signal peptides [Gascuel and Danchin, 1988], and grammatical inference techniques to study *E. coli* promoter sequences [Park and Huntsberger, 1990] and 5'-splice sites in eukaryotic mRNAs [Kudo et al., 1987].

9 Conclusion

> *"Precisely constructed models for linguistic structure can play an important role, both negative and positive, in the process of discovery itself. By pushing a precise but inadequate formulation to an unacceptable conclusion, we can often expose the exact nature of this inadequacy and, consequently, gain a deeper understanding of the linguistic data. More positively, a formalized theory may automatically provide solutions for many problems other than those for which it was explicitly designed." [Chomsky, 1957]*

Treating genes and potentially entire genomes as languages holds great appeal in part because it raises the possibility of producing concise generalizations about the information contained in biological sequences and how it is "packaged". One hopes that grammars used for this purpose would comprise a model of some underlying physical objects and processes, and that grammars may in fact serve as an appropriate tool for theory formation and testing, in the linguistic tradition. This article has suggested a number of ways in which this might occur, many of which are summarized below:

Parse trees may reflect secondary structure. It is considered a virtue of natural language grammars for their parse trees to capture schematically some inherent structure of a sentence (see, for example [Gazdar, 1985]). The reader is invited to draw parse trees from some of the orthodox secondary structure grammars of section 2.4, and observe how remarkably their outlines conform to the actual physical structures described. As we have noted, this extends also to alternative secondary structures, modelled by ambiguous grammars.

Grammar nonterminals might model biochemical entities. For example, nonterminals representing DNA-binding proteins could "rewrite" as the appropriate consensus binding sites, which would be especially useful in cases where proteins or protein complexes bind several sites and bring them into physical proximity as a result. Such complexes, and indeed other "layered" protein-protein interactions as well (e.g. the complement cascade in immunology [Watson et al., 1987]), could also be modelled hierarchically by grammars.

Grammar rules could describe intra-molecular interactions. The nonterminals in secondary structure grammars can be viewed as representing the hydrogen bonding between complementary bases. Generalizing this, other forms of chemical interaction or dependencies between distant sites in a macromolecule could be modelled, as suggested above for protein structure. Just as parse trees can depict secondary structure, more complex structures might be specified quite literally using grammar formalisms from the field of syntactic pattern recognition in which terminals are actually two- or three-dimensional subgraphs connected by derivation [Fu, 1982].

Greater-than-context-free grammars can model mutation and evolution. As was seen in section 2.8.2, rules producing side-effects on the input string can capture these processes, and the ability of such grammars to take account of lexical elements (i.e. "contexts") that could control such processes is particularly attractive. *Transformational grammar,* subsequently developed by Chomsky to account for, among other things, "movement" phenomena in natural language, might also be useful in describing the allowable variations on the *deep structure*, or canonical syntax, of a gene (together, perhaps, with its regulatory elements [Collado-Vides, 1989a,b]).

Grammar derivation could model gene expression. The notion of successful derivation from a gene grammar being analogous to the expression of that gene in the cell was discussed at length in section 2.7. In this model, nonterminals thus represent gene products and their component parts, or, in the context of gene regulation, the aggregate of

lexical and "environmental" elements required to accomplish expression at a given time and place.

Parsing might mimic certain biochemical processes. Such environmental elements of gene expression are of course problematic, but we have suggested elsewhere how parsing might yet be a suitable simulation tool for these control systems [Searls, 1988]. It is interesting to note how recursive rules in a left-to right parser resemble the physical action of certain *processive* enzymes that travel along nucleic acid molecules, even in some cases performing a kind of "backtracking" error correction. The suitability of even more basic language-theoretic operations for depicting biological processes like replication and recombination was noted in section 2.5.

Besides the potential role of linguistic tools in modelling of biological systems, we have also discussed at length the use of grammars for specification (both of sequence elements and of algorithms), pattern-matching search, and as abstractions that could lead to insights about the organization of genetic information and its tractability to computational approaches. It seems clear that the detection and analysis of genes and other features of the genome could benefit from parser technology and a general awareness of the linguistic properties of the domain. The notion of a comprehensive grammar describing the genome or even individual genes in their full generality is clearly quixotic, but the effort to approach this ideal may yet afford a better understanding of what is surely a fundamentally linguistic domain.

Acknowledgements

The author wishes to thank Drs. Lynette Hirschman and Chris Overton for careful reading of early drafts and, along with Drs. Erik Cheever, Rebecca Passonneau, and Carl Weir, for many helpful discussions. The author is also grateful to Dr. Ross Overbeek and his student Ron Taylor, and to Drs. Mick Noordewier and Jude Shavlik, for acquainting him with their work in progress. Invaluable comments and suggestions were provided by Jim Tisdall in the area of formal language theory. A special debt is owed to Dr. Allen Sears for his enthusiastic patronage of these efforts. This work was supported in part by the Department of Energy, Office of Energy Research, under genome grant number DE-FG02-90ER60998.

Note

* Abbreviations used: BNF: Backus-Naur form; CFL: context-free language; CSL: context-sensitive language; CKY: Cocke-Kasami-Younger (parsing algorithm); DCG: definite clause grammar; FSA: finite state au-

tomaton; GSM: generalized sequential machine; IL: indexed language; L-system: Lindenmayer system; PDA: pushdown automaton; RL: regular language; SVG: string variable grammar.

Bibliography

J. P. Abrahams, M. van den Berg, E. van Batenburg, and C. Pleij. Prediction of RNA Secondary Structure, Including Pseudoknotting, by Computer Simulation. *Nucleic Acids Res.* 18:3035-3044, 1990.

A. V. Aho and T. G. Peterson. A Minimum Distance Error-correcting Parser for Context-free Languages. *SIAM J. Comput.* 1(4):305-312, 1972.

J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Comm. of the ACM* 26:832-843, 1983.

J. F. Allen. *Natural Language Understanding*. Benjamin/Cummings, Menlo Park, 1987.

V. Brendel and H. G. Busse. Genome Structure Described by Formal Languages. *Nucleic Acids Res.* 12:2561-2568, 1984.

V. Brendel, J. S. Beckmann, and E. N. Trifinov. Linguistics of Nucleotide Sequences: Morphology and Comparison of Vocabularies. *J. Biomol. Struct. Dynamics* 4:11-21, 1986.

J. Bresnan, R. Kaplan, S. Peters, and A. Zaenen. Cross-serial Dependencies in Dutch. *Linguistic Inquiry* 13:613-636, 1982.

E. A. Cheever, G. C. Overton, and D. B. Searls. Fast Fourier Transform-based Correlation of DNA Sequences Using Complex Plane Encoding. *Comput. Applic. Biosci.* 7(2)143-159, 1991.

N. Chomsky. *The Logical Structure of Linguistic Theory.* The University of Chicago Press, Chicago (1975), 1955.

N. Chomsky. *Syntactic Structures.* Mouton, The Hague, 1957.

N. Chomsky. On Certain Formal Properties of Grammars. *Informat. Control* 2:137-167, 1959.

N. Chomsky. Formal Properties of Grammars. In D. Luce, R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology II.* John Wiley & Sons, New York, 1963.

N. Chomsky and M. P. Schutzenberger. The Algebraic Theory of Context-free Languages. In P. Braffort and D. Hirschberg, editors, *Computer Program–ming and Formal Systems*, pp. 118–161. North-Holland, Amsterdam, 1963.

N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, 1965.

J. Collado-Vides. A Transformational-grammar Approach to the Study of the Regulation of Gene Expression. *J. Theor. Biol.* 136:403-425, 1989a.

J. Collado-Vides. Towards a Grammatical Paradigm for the Study of the Regulation of Gene Expression. In B. Goodwin and P. Saunders, editors, *Theoretical Biology: Epigenetic and Evolutionary Order*, pages 211-224. Edinburgh University Press, 1989b.

J. Collado-Vides. A Syntactic Representation of Units of Genetic Information. *J. Theor. Biol.* 148:401-429, 1991a.

J. Collado-Vides. The Search for a Grammatical Theory of Gene Regulation Is Formally Justified by Showing the Inadequacy of Context-free Frammars. *Comput. Applic. Biosci.* 7(3):321-326, 1991b.

W. Ebeling and M. A. Jimenez-Montano. On Grammars, Complexity, and Information Mea-

sures of Biological Macromolecules. *Math. Biosci.* 52:53-71, 1980.

K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Englewood Cliffs, 1982.

O. Gascuel and A. Danchin. Data Analysis Using a Learning Program, a Case Study: An Application of PLAGE to a Biological Sequence Analysis. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 390-395, 1988.

D. Gautheret, F. Major, and R. Cedergren. Pattern Searching/Alignment with RNA Primary and Secondary Structures: An Effective Descriptor for tRNA. *Comput. Applic. Biosci.* 6(4):325-331, 1990.

G. Gazdar. Applicability of Indexed Grammars to Natural Languages. CSLI-85-34, Center for the Study of Language and Information, Stanford, 1985.

M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, 1978.

T. Head. Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors. *Bull. math. Biol.* 49(6):737-759, 1987.

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

M. A. Jimenez-Montano. On the Syntactic Structure of Protein Sequences and the Concept of Grammar Complexity. *Bull. math. Biol.* 46(4):641-659, 1984.

M. M. Konarska and P. A. Sharp. Structure of RNAs Replicated by the DNA-dependent T7 RNA polymerase. *Cell* 63:609-618, 1990.

M. Kudo, Y. Iida, and M. Shimbo. Syntactic Pattern Analysis of 5'-splice Site Sequences of mRNA Precursors in Higher Eukaryotic Genes. *Comput. Applic. Biosci.* 3(4):319-324, 1987.

R. H. Lathrop, T. A. Webster, and T. F. Smith. Ariadne: Pattern-directed Inference and Hierarchical Abstraction in Protein Structure Recognition. *Comm. of the ACM* 30:909-921, 1987.

B. Lewin. *Genes*. John Wiley & Sons, Inc., New York, third edition, 1987.

A. Lindenmayer. Mathematical Models for Cellular Interaction in Develop–ment. *J. Theor. Biol.* 18:280-315, 1968.

G. C. Overton and J. A. Pastor. A Platform for Applying Multiple Machine-learning Strategies to the Task of Understanding Gene Structure. In *Proceedings of the 7th Conference on Artificial Intelligence Applications*, pages 450-457. IEEE, 1991.

K. Park and T. L. Huntsberger. Inference of Context-free Grammars for Syntactic Analysis of DNA Sequences. In *AAAI Spring Symposium Series*, Stanford, 1990. American Association for Artificial Intelligence.

J. A. Pastor, K. Koile, and G. C. Overton. Using Analogy to Predict Functional Regions on Genes. In *Proceedings of the 24th Hawaii International Conference on System Science*, pages 615-625, 1991.

S. Pietrokovski, J. Hirshon, and E. N. Trifinov. Linguistic Measure of Taxonomic and Functional Relatedness of Nucleotide Sequences. *J. Biomol. Struct. and Dyn.* 7(6):1251-1268, 1990.

F. C. N. Pereira and S. M. Shieber. *Prolog and Natural-Language Analysis*. Center for the Study of Language and Information, Stanford, 1987.

F. C. N. Pereira and D. H. D. Warren. Definite Clause Grammars for Language Analysis. *Artif. Intell.* 13:231-278, 1980.

P. A. Pevzner, M. Y. Borodovsky, and A. A. Mironov. Linguistics of Nucleotide Sequences: I. The Significance of Deviation from Mean Statistical Characteristics and Prediction of the Fre-

quency of Occurence of Words. *J. Biomol. Struct. and Dyn.*, 6:1013-1026, 1989a.

P. A. Pevzner, M. Y. Borodovsky, and A. A. Mironov. Linguistics of Nucleotide Sequences: II. Stationary Words in Genetic Texts and Zonal Structure of DNA. *J. Biomol. Struct. and Dyn.*, 6:1027-1038, 1989b.

C. W. A. Pleij. Pseudoknots: A New Motif in the RNA Game. *Trends in the Biosci.* 15:143-147, 1990.

P. M. Postal and D. T. Langendoen. English and the Class of Context-free Languages. *Computational Linguistics* 10:177-181, 187-188, 1984.

P. Prusinkiewicz and J. Hanan. *Lindenmayer Systems, Fractals, and Plants*, volume 79 of *Lecture Notes in Biomathematics.* Springer-Verlag, New York, 1989.

S. M. Schieber. Evidence Against the Context-freeness of Natural Language. *Linguistics and Philosophy* 8:333-343, 1985.

J. L. Schroeder and F. R. Blattner. Formal Description of a DNA Oriented Computer Language. *Nucleic Acids Res.* 10:69, 1982.

D. B. Searls. Representing Genetic Information with Formal Grammars. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 386-391. American Association for Artificial Intelligence, 1988.

D. B. Searls. Investigating the Linguistics of DNA with Definite Clause Grammars. In E. Lusk and R. Overbeek, editors, *Logic Programming: Proceedings of the North American Conference on Logic Programming*, volume 1, pages 189-208. Association for Logic Programming, 1989a.

D. B. Searls. Signal Processing with Logic Grammars. *Intelligent Systems Rev.* 1(4):67-88, 1989b.

D. B. Searls and S. Liebowitz. Logic Grammars as a Vehicle for Syntactic Pattern Recognition. In *Proceedings of the Workshop on Syntactic and Structural Pattern Recognition*, pages 402-422. International Association for Pattern Recognition, 1990.

D. B. Searls and M. O. Noordewier. Pattern-matching Search of DNA Sequences Using Logic Grammars. In *Proceedings of the 7th Conference on Artificial Intelligence Applications*, pages 3-9. IEEE, 1991.

P. H. Sellers. On the Theory and Computation of Evolutionary Distances. *SIAM J. Appl. Math.* 26:787-793, 1974.

B. Shanon. The Genetic Code and Human Language. *Synthese* 39:401-415, 1978.

T. A. Sudkamp. *Languages and Machines.* Addison-Wesley, Reading, 1988.

J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner. *Molecular Biology of the Gene*. Benjamin/Cummings, Menlo Park, 1987.