CONF-9507246--5

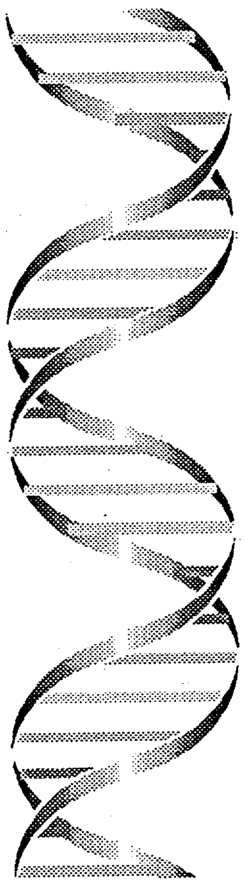
ISMB-95
ROBINSON COLLEGE,
CAMBRIDGE

Tutorial Programme
Sunday 15 July 1995

TUTORIAL T5

The Computational Linguistics of Biological Sequences

(David Searls)



ISMB'95 Tutorial Notes

The Computational Linguistics of Biological Sequences

David Searls

Departments of Genetics and
Computer and Information Science
University of Pennsylvania
Clinical Research Building, Rm. 475
422 Curie Boulevard
Philadelphia, PA 19104-61445
dsearls@cbil.humgen.upenn.edu

1 Background

Up to now, formal language theory and computational linguistics have dealt primarily with natural human languages, artificial computer languages, and little else in the way of serious applications. However, because of rapid advances in the field of molecular biology it now appears that biological sequences such as DNA and protein, which are after all composed quite literally of sets of strings over well-defined chemical alphabets, may well become the third major domain of the tools and techniques of mathematical and computational linguistics. The work of the author [6, 7, 8, 9, 10, 11] and a number of others [1, 2, 3, 4] has served to establish the "linguistic" character of biological sequences from a number of formal and practical perspectives, while at the same time the international effort to map and sequence the human genome is producing data at a prodigious rate. Not only does this data promise to provide a substantial corpus for further development of the linguistic theory of DNA, but its enormous quantity and variety may demand just such an analytic approach, with computational assistance, for its full understanding.

The language of DNA, consisting of strings over the four-letter alphabet of the nucleotide *bases* 'a', 'c', 'g', and 't', is distinguished first of all by the sizes of those strings. The human genome contains 24 distinct types of chromosomes, each in turn containing double helices of DNA, with lengths totalling over three billion bases. Scattered among the chromosomes are genes which can extend over tens of thousands of bases, and which are arguably the "sentences" of the genetic language, possessing as they do extensive substructure of their own [8]. Moreover, genes and similar high-level features occur in a wide range of forms, with arrangements of "words" of base sequences seemingly as varied as those in natural language. Clearly any attempt to specify and perhaps to parse such features must deal first and foremost with the sheer magnitude of the language, in terms of both lengths of strings and cardinality. However, there are other, more subtle challenges, having to do with the nature of the strings to be described. Some of these features of the language, around which the author has been developing grammatical formalisms and practical domain-specific parsers, are described in this and the following section. The reader may find additional biological detail in any standard textbook of molecular biology (e.g. [5, 14], or the more concise [13]).

DNA is a double-stranded molecule, with the strands possessing an opposite directionality; the bases that lie across from each other in the two strands pair in a complementary fashion, i.e. 'g' pairs with 'c' and vice versa, and 'a' pairs with 't' and vice versa. Inverting a substring of DNA actually requires not only that a double-stranded segment be excised and reversed, but that the opposite, complementary strands be rejoined, to maintain the proper directionality. The result is that in the reversed string each base is replaced by its complement.

The biological "operation" of inversion is just one of many types of mutation to which DNA is subject, in the course of evolution; others include deletion, insertion, and transposition, in addition to simple *point* mutations involving substitution of bases. One of the most important operations is duplication, which in fact is a central mechanism of molecular evolution: a substring is duplicated, and then the copies may evolve apart by further mutation until they assume different functions. This has several important consequences. First, it suggests that duplication will be an important feature of the language of DNA. Second, it indicates that features of a similar nature can vary as a consequence of mutation, and indeed approximate matching will prove to be an important factor. Third, it suggests that features might exhibit movement phenomena, perhaps reminiscent of natural language, and again this is borne out by observation: regulatory signals, in particular, exhibit a degree of "free word order" in their relative placements.

DNA is also noteworthy for the large degree of interleaving and even overlap in the information it encodes. The business of a gene is actually to be *transcribed* to another (similar) type of molecule called RNA, which has its own language determining how it can fold up into *secondary structure* and how it is further processed by internal deletion ("*splicing*") or other forms of editing. RNA, in turn, is most often systematically *translated* to protein, which has a vastly different alphabet and functional repertoire. While DNA has its own signals that determine operations performed directly on it in the nucleus of the cell, it also contains within the same regions the encoded sequences of RNA and protein and the signals necessary for their processing and functioning at different times in other parts of the cell. This overloading of the language of DNA can go to extremes, for instance in cases where more than one protein is encoded in literally overlapping DNA sequence.

These various transformations of the information in DNA can be modeled by simple finite-state *transducers*, and in fact such transducers can be used to model mutation as well. In a paper in this conference [12], we show how such transducers can be used to derive algorithms that are characteristically used to find the best alignment of strings that are thought to arise from a common ancestor.

Another general characteristic of much DNA is the relative sparseness of its information content. Genes comprise only a few percent of many genomes, and the vast tracts between genes, though they may contain important regulatory regions or establish global properties, are almost certainly expendable in some degree. Even genes themselves are interrupted by long sequences called *introns* that do not encode anything essential to the final protein gene product, and are in fact spliced out of the corresponding RNA.

Finally, it should be borne in mind that the strings of these biological languages are literal, physical objects. In particular, they interact not only with their environment (including DNA-binding proteins that recognize specific "words"), and with other strings (as in the double helix of DNA), but also with themselves (as in RNA secondary structure). In the latter case, the RNA actually bends back upon itself and base pairs as if it were the two halves of a double helix; this in fact occurs at biological palindromes of the sort described above, for reasons that may be apparent.

Such structures can become quite complex and highly branched, producing not only palindromic regions but additional forms of non-context free phenomena, and showing evidence of a purposeful *ambiguity* in the sense that multiple structures arise from the same sequence of bases [8, 9]. Such interactions between elements of a string folding back on itself form natural dependencies, which we might well wish to capture using appropriate grammar formalisms.

While this tutorial will concentrate on nucleic acid sequences because of their relative simplicity, it should be borne in mind that protein sequences are analogous in many respects, particularly their folding behavior. Proteins have a much richer variety of interactions, but in theory the same linguistic principles could come to bear in describing dependencies between distant residues that arise by virtue of three-dimensional structure.

References

[1] 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.

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

[3] J. Collado-Vides. The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars. *CABIOS*, 7(3):321–326, 1991.

[4] 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.

[5] B. Lewin. *Genes V*. Oxford University Press, Oxford, UK, 1994.

[6] D. B. Searls. Representing genetic information with formal grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 386–391. American Association for Artificial Intelligence, 1988.

[7] 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*, pages 189–208. MIT Press, 1989.

[8] D. B. Searls. The linguistics of DNA. *American Scientist*, 80(6):579–591, 1992.

[9] D. B. Searls. The computational linguistics of biological sequences. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, chapter 2, pages 47–120. AAAI Press, 1993.

[10] D. B. Searls and S. Dong. A syntactic pattern recognition system for DNA sequences. In H. A. Lim, J. Fickett, C. R. Cantor, and R. J. Robbins, editors, *Proceedings of*

the 2nd International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis, pages 89–101. World Scientific, 1993.

[11] D. B. Searls and M. O. Noordewier. Pattern-matching search of DNA sequences using logic grammars. In *Proceedings of the Conference on Artificial Intelligence Applications*, pages 3–9. IEEE, 1991.

[12] D.B. Searls and K. Murphy. Automata-theoretic models of mutation and alignment. *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 1995.

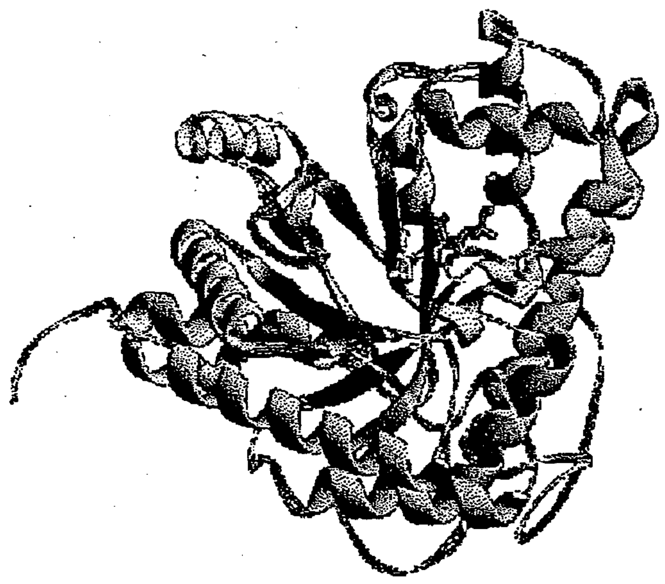
[13] J.D. Watson, M. Gilman, J. Witkowski, and M. Zoller. *Recombinant DNA*. Scientific American Books, New York, NY, 1992.

[14] 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, CA, 1987.

Proteins (I)

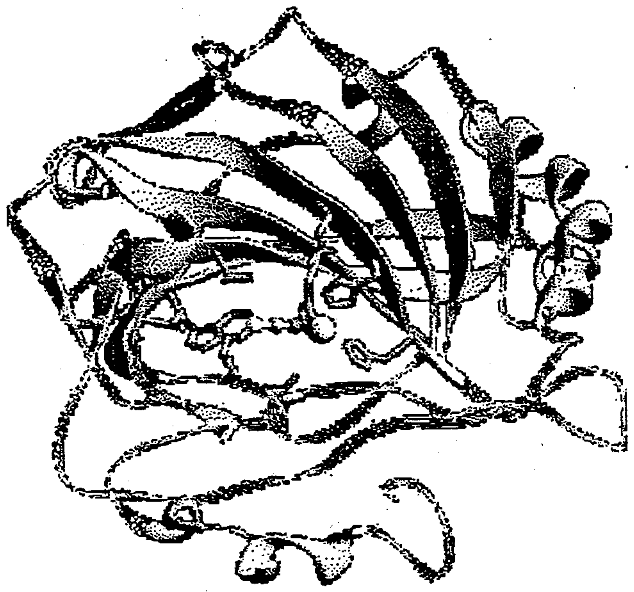
- proteins are polymers of amino acids, a 20 letter alphabet often written as single uppercase letters
- amino acids have side chains that vary greatly in terms of properties such as charge, bulk, hydrophobicity (i.e. tendency to avoid water), etc.
- proteins assume a 3-dimensional shape called their tertiary structure, which is a function of their primary structure or sequence
- the tertiary structure of a protein determines its biological function, e.g. by forming a specific shape to fit another molecule, bind to it, and catalyze an enzymatic reaction within an active site

Proteins (II)



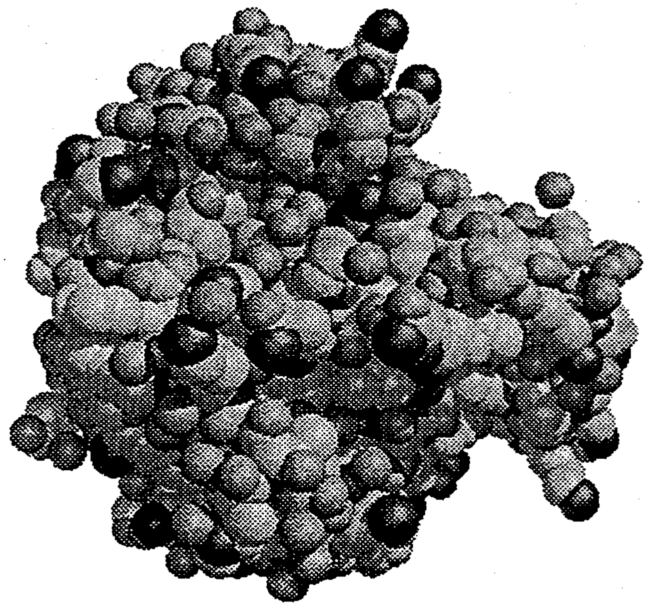
- within the 3D shapes certain substructures may recur, such as alpha helices and beta sheets, as well as regions of random coil
- these recurring motifs are called the secondary structure

Proteins (III)



- secondary structural elements interact with each other in a variety of ways, e.g. hydrogen bonds between beta sheets (either parallel or anti-parallel), charge interactions, hydrophobicity, etc.

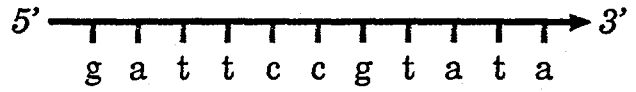
Proteins (IV)



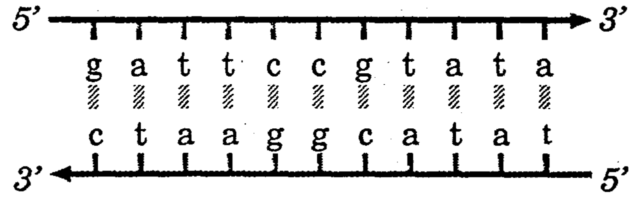
- although proteins (and nucleic acids) are viewed as strings of symbols, recognize that they are folded "blobs" in actuality
- a space-filling model of the last example (carbonic anhydrase) shows the electron clouds

DNA Structure (I)

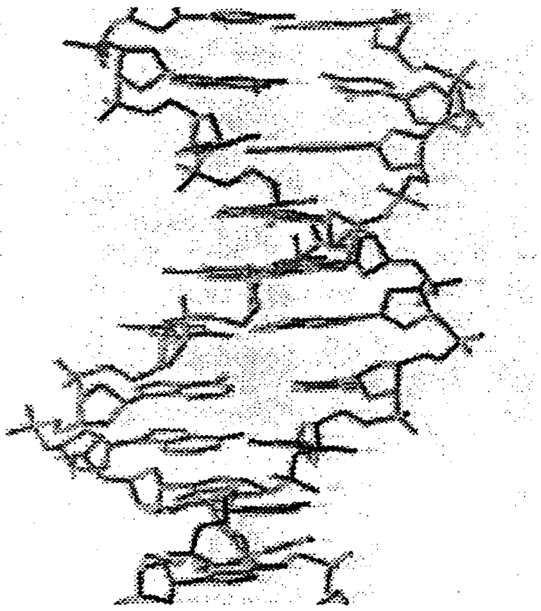
- DNA consists of nucleotide bases g, a, t, and c, strung together on a directional sugar-phosphate backbone



- molecules of DNA normally exist in anti-parallel pairs, bound by complementarity between bases a/t and g/c



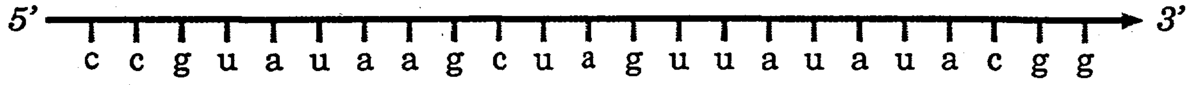
DNA Structure (II)



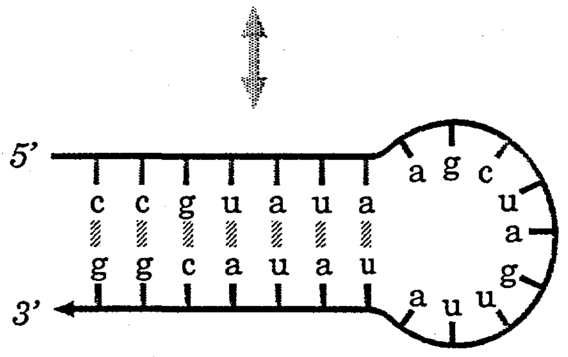
- the strands coil around each other in a right-handed double helix, the bases on the interior stacked like dishes
- the bases pair only via hydrogen bonds, so can be separated by heat
- g/c bonds are stronger than a/t, so have higher melting temperature

RNA Structure

- RNA is chemically similar to DNA, but has a slightly different alphabet (g, a, u, c) and is single-stranded

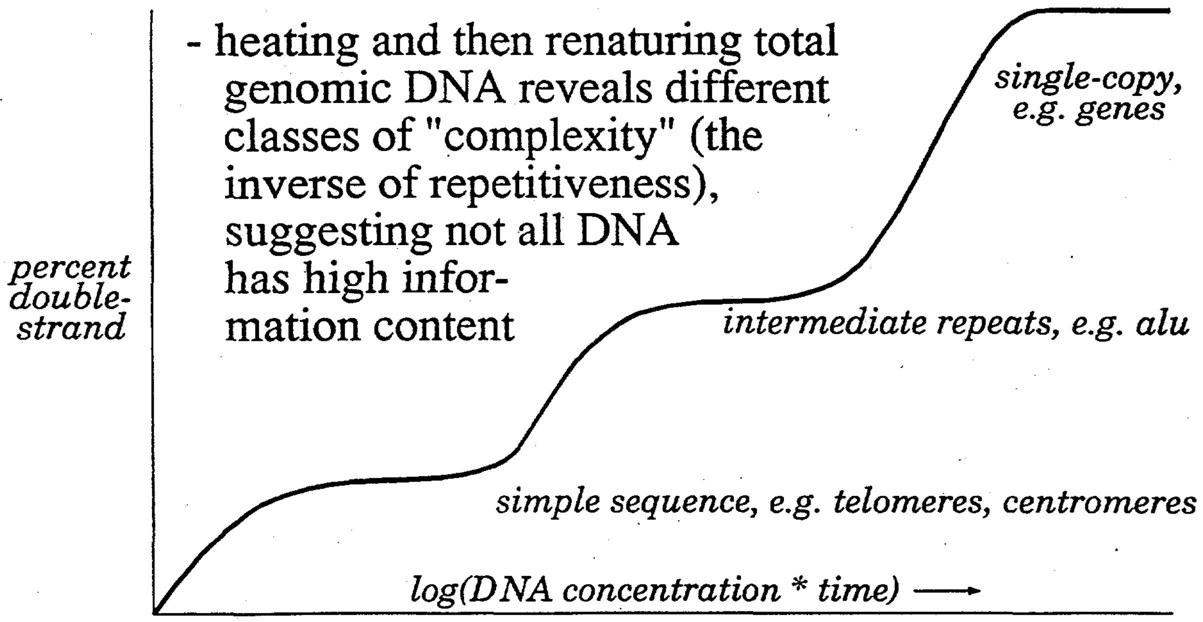


- however, RNA is able to fold back on itself to form secondary structure, e.g. a simple stem-and-loop
- base-pairing need not be perfect; the structure depends on thermodynamics



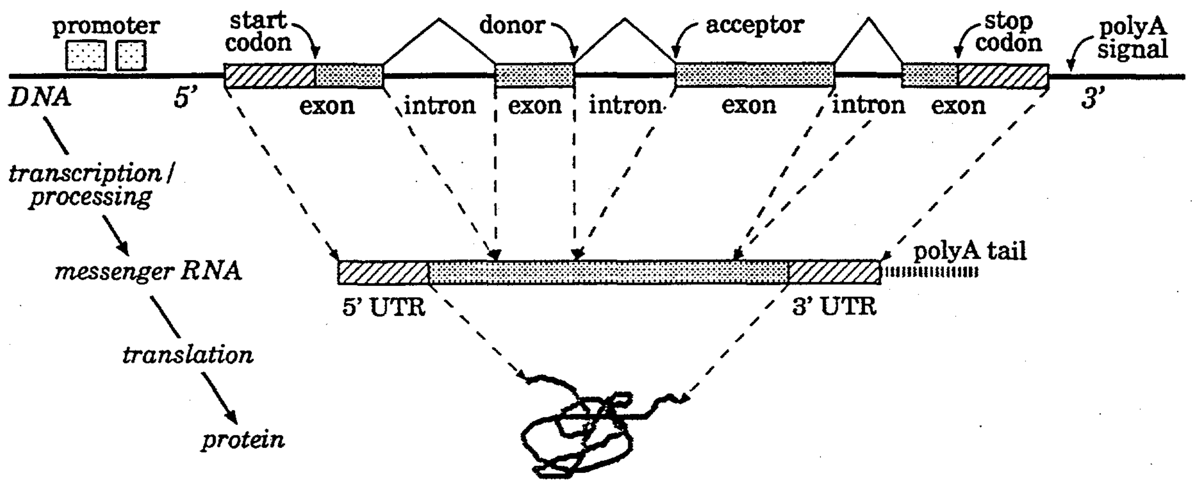
Hybridization

- heating and then renaturing total genomic DNA reveals different classes of "complexity" (the inverse of repetitiveness), suggesting not all DNA has high information content



Gene Structure

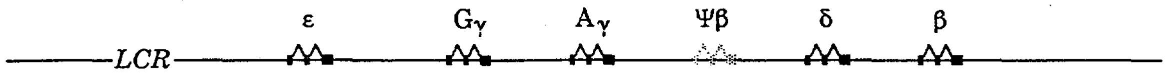
- eukaryotic genes are interrupted by sequences that are spliced out in the process of gene expression



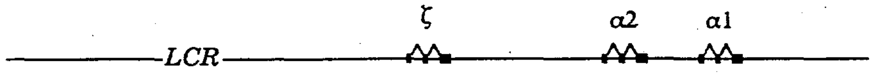
Gene Distribution

- genes only constitute a few percent of the genome, and are sometimes distributed in related families with coordinate control at different times and places in development, etc.

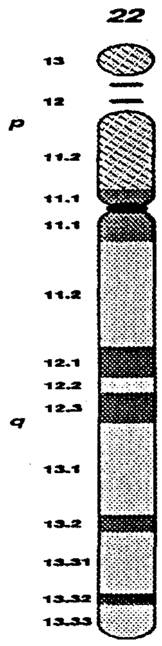
Beta Hemoglobin Gene Cluster (~70kb)



Alpha Hemoglobin Gene Cluster (~40kb)



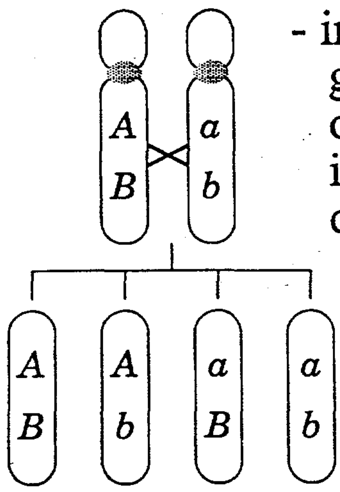
Chromosomes



- in higher organisms DNA is packaged into chromosomes, where it is highly coiled and supercoiled, and complexed with proteins
- in diploid organisms, chromosomes occur in pairs (with the exception of the sex chromosomes); humans have 24 distinct chromosomes
- genetic variation arises by mutations of the DNA, and by recombination between the paired (but non-identical) chromosomes
- the entire complement of DNA in an organism is called the genome, and it comprises about 3 billion bases in humans

Classical Genetics

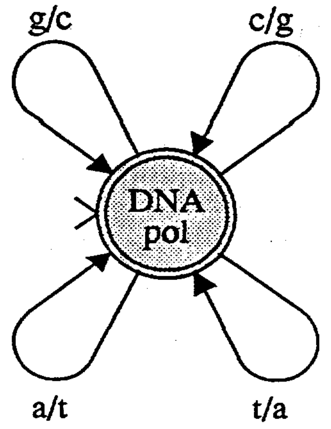
- traits are encoded in the DNA as genes, which vary among individuals; different "versions" of a genetic locus are called alleles, and the phenomenon is called polymorphism



- in classical genetics the distance between genes was measured in terms of frequency of recombinations between them, and this is still one of the most important tools for discovering the location of diseases, etc.
- there are currently estimated to be about 100,000 genes in the human genome, and a polymorphism every 100-200 bases on average, yet only ca. 1% difference from chimpanzee

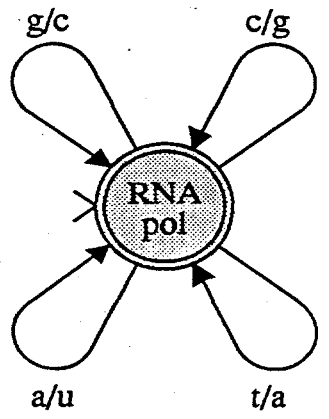
Replication

- replication of DNA is possible because of the specificity of base-pairing, allowing enzymes called DNA polymerases to synthesize a new strand from a template, in a 5' to 3' direction
- polymerases are mostly processive, i.e. they move along the template synthesizing the new strand; they can be seen as finite transducers:
- polymerases require a specific primer to hybridize and initiate the reaction, plus bases to add



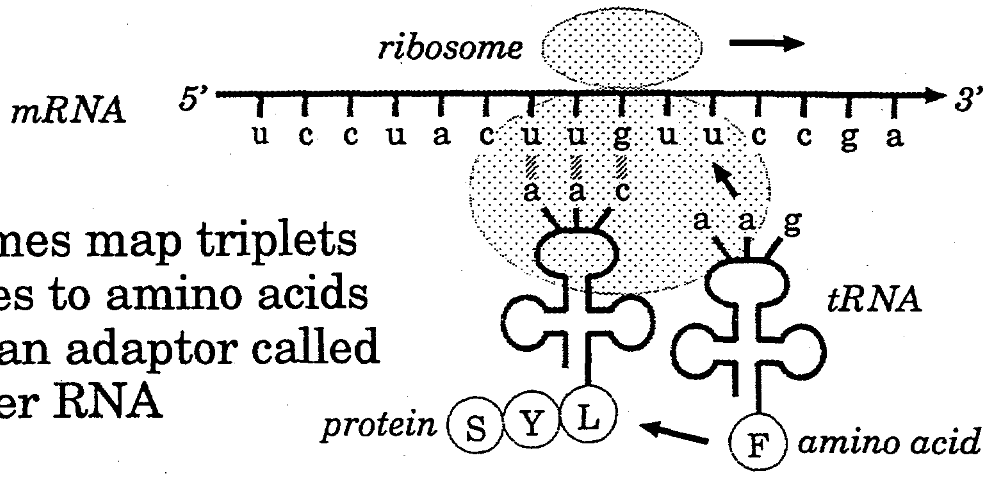
Transcription

- RNA polymerases mediate the transcription of RNA from a DNA template, generally under the influence of transcription factors that recognize specific start signals in the DNA
- RNA polymerases also synthesize 5' to 3', but from one strand only
- reverse transcriptases can copy from RNA back to DNA, and are made by tumor viruses; they are an important tool for creating cDNA libraries from mRNA



Translation

- translation from RNA to protein involves a more complex cellular machine called a ribosome



- ribosomes map triplets of bases to amino acids using an adaptor called transfer RNA

The Genetic Code

- the mapping from triplets or codons to amino acids lacks three cases: the stop codons that end proteins

Second Position

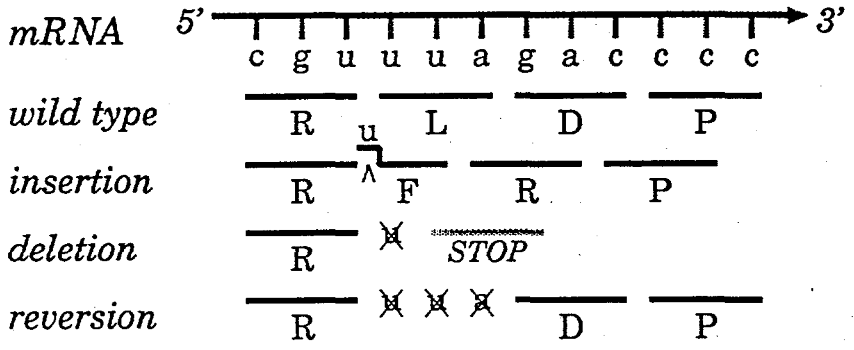
First Position

	u		c		a		g		
u	uuu	Phe (F)	ucu	Ser (S)	uau	Tyr (Y)	ugu	Cys (C)	u
	uuc	"	ucc	"	uac	"	ugc	"	c
	uua	Leu (L)	uca	"	uaa	STOP	uga	STOP	a
	uug	"	ucg	"	uag	STOP	ugg	Trp (W)	g
c	cuu	Leu (L)	ccu	Pro (P)	cau	His (H)	cgu	Arg (R)	u
	cuc	"	ccc	"	cac	"	cgc		c
	cua	"	cca	"	caa	Gln (Q)	cga	"	a
	cug	"	ccg	"	cag	"	cgg	"	g
a	auu	Ile (I)	acu	Thr (T)	aau	Asn (N)	agu	Ser (S)	u
	auc	"	acc	"	aac	"	agc	"	c
	aua	"	aca	"	aaa	Lys (K)	aga	Arg (R)	a
	aug	Met (M)	acg	"	aag	"	agg	"	g
g	guu	Val (V)	gcu	Ala (A)	gau	Asp (D)	ggu	Gly (G)	u
	guc	"	gcc	"	gac	"	ggc	"	c
	gua	"	gca	"	gaa	Glu (E)	gga	"	a
	gug	"	gcg	"	gag	"	ggg	"	g

Third Position

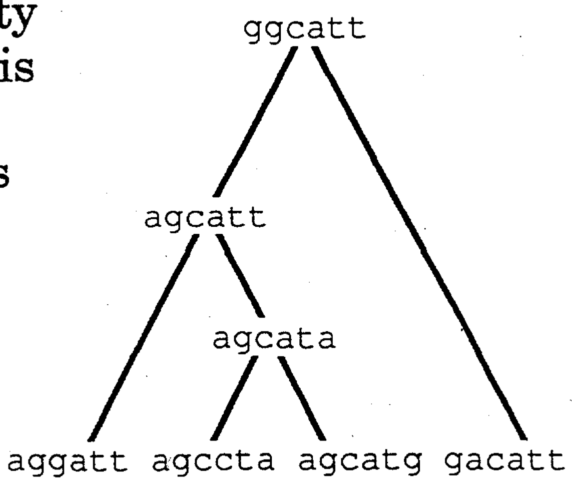
Mutation

- substitution of bases may have little effect (e.g. at the 3rd codon position, or if a similar amino acid is substituted) or may be fatal (e.g. creating a stop)
- insertions and deletions (indels) may or may not cause frameshift mutations, usually fatal



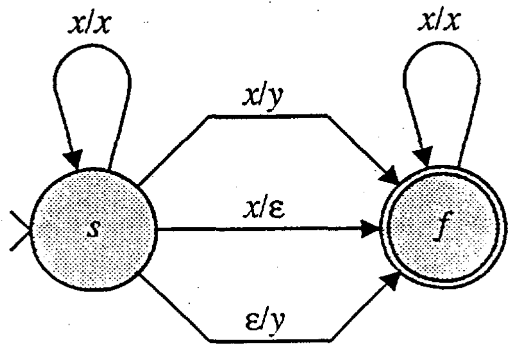
Evolution

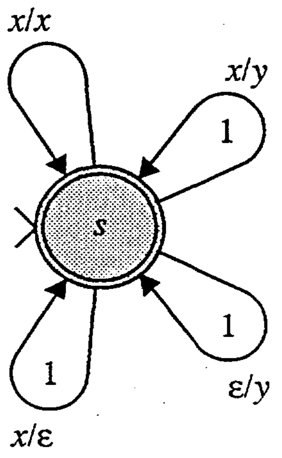
- evolution proceeds primarily by duplication of genes followed by divergence of function through mutation
- the most important activity in computational biology is the detection of distant similarities or homologies among present-day sequences
- construction of the most parsimonious phylogeny sheds light on evolution



Edit Distance

- a finite transducer that models mutation can be minimum-distance editor





$$D[i,j] = \min \begin{cases} D[i-1,j-1] + \delta_{i,j} & \text{where } \delta_{i,j} = 0 \text{ if } x_i = y_j \text{ else } \delta_{i,j} = 1 \\ D[i-1,j] + 1 & \\ D[i,j-1] + 1 & D[i,0] = i \quad D[0,j] = j \end{cases}$$

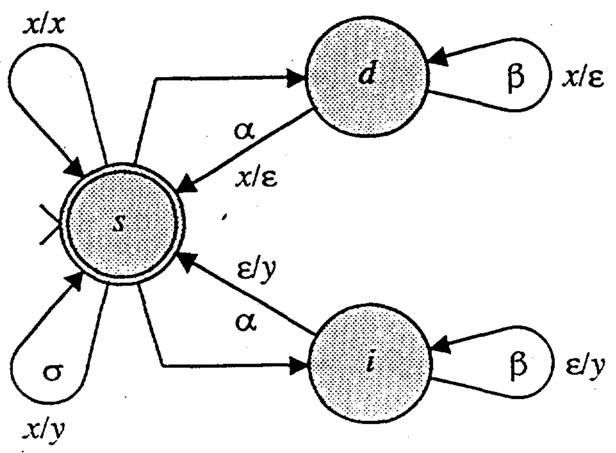
Dynamic Programming

- dynamic programming finds edit distance in O(mn)
- each position in the matrix only depends on its neighbors to the left, above, and diagonally, to align the inputs

```
-TCGGAGTCA
 || ||  |||
ATCTGA-TCA
```

		T	C	G	G	A	G	T	C	A
	0	1	2	3	4	5	6	7	8	9
A	1	1	2	3	4	4	5	6	7	8
T	2	1	2	3	4	5	5	5	6	7
C	3	2	1	2	3	4	5	6	5	6
T	4	3	2	2	3	4	5	5	6	6
G	5	4	3	2	2	3	4	5	6	7
A	6	5	4	3	3	2	3	4	5	6
T	7	6	5	4	4	3	3	3	4	5
C	8	7	6	5	5	4	4	4	3	4
A	9	8	7	6	6	5	5	5	4	3

Affine Gaps



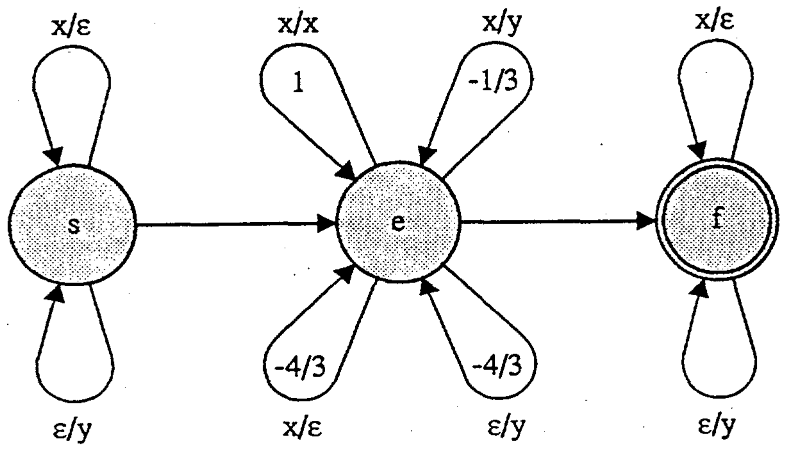
- a more realistic model of indels treats each gap as a single event, with a large initial cost and then a smaller incremental one to enlarge it [Gotoh]

$$s[a,b] = \min \begin{cases} s[a-1,b-1] & \text{if } x_a = y_b \\ s[a-1,b-1]+\sigma & \text{if } x_a \neq y_b \\ d[a,b] & \\ i[a,b] & s[0,0]=0 \end{cases}$$

$$d[a,b] = \min \begin{cases} s[a-1,b]+\alpha \\ d[a-1,b]+\beta \end{cases}$$

$$i[a,b] = \min \begin{cases} s[a,b-1]+\alpha \\ i[a,b-1]+\beta \end{cases}$$

Local Alignment



- modern search algorithms use similarity, not distance, and find short local alignments [Smith & Waterman]

$$s[a,b] = \max \begin{cases} s[a-1,b] \\ s[a,b-1] \\ e[a,b] \end{cases} = \max_{\substack{0 \le i \le a \\ 0 \le j \le b}} e[i,j]$$

$$f[a,b] = \max \begin{cases} f[a-1,b] \\ f[a,b-1] \end{cases} = 0$$

$$e[a,b] = \max \begin{cases} e[a-1,b-1]+1 & \text{if } x_a = y_b \\ e[a-1,b-1]-1/3 & \text{if } x_a \neq y_b \\ e[a-1,b]-4/3 & \\ e[a,b-1]-4/3 & \\ f[a,b]\ (=0) & \end{cases}$$

2 Theory

In the realm of formal computational linguistics, a language is defined in terms of an *alphabet* Σ, which is a finite set of *symbols*; in the case of DNA sequences, such symbols should be the nucleotide bases, so that we have $\Sigma_{\text{DNA}} = \{\text{g}, \text{c}, \text{a}, \text{t}\}$. A DNA molecule can then be represented as a *string* over Σ_{DNA}, that is, a finite sequence of symbols from Σ_{DNA}. The set of all possible strings over an alphabet is denoted by Σ^*, and a *language*, formally, is any subset of Σ^*.

The concern of formal linguistics is the finite representation of languages which may themselves be infinite; the goal is an economy of expression, in an abstract representation, as an alternative to exhaustively enumerating all the allowable strings in a language. Such cogency may also have the benefit of capturing some kind of essential, clarifying generalization about the structure or *syntax* of a linguistic system, preferably related to the meaning or *semantics* of the language elements. For this purpose, language generators called *grammars* have proven extremely useful. Grammars specify languages through sets of *rules* or *productions*, which achieve the desired succinctness largely by referring to each other and to themselves *recursively*. Perhaps the most important class of grammars is the *context-free grammars* (CFGs, which specify the *context-free languages*, CFLs). A CFG has the set Σ of symbols from the language, called *terminals*, and an additional set of symbols called *nonterminals*; these symbols are used in a finite set of rules whose members are denoted by $A \rightarrow u$ where A is a nonterminal and u is a string of terminals and nonterminals. A grammar generates the string elements of its language by taking a starting symbol S and rewriting it, by repeatedly finding a rule whose left-hand side matches some nonterminal in the current string, and substituting that rule's right-hand side, until the string contains all terminals. Such derivation steps are denoted by a double arrow, $\Longrightarrow$, so that for the simple grammar with $\Sigma = \{a, b, c\}$, nonterminals S and X, and rules $S \rightarrow aX$, $X \rightarrow bX$, and $X \rightarrow c$, one possible derivation is:

$$S \Longrightarrow aX \Longrightarrow abX \Longrightarrow abbX \Longrightarrow abbc$$

We can say that the language generated by a grammar G is the set of all strings w over Σ such that w is derivable from S, or in set notation $\{w \in \Sigma^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^*$ denotes any number of applications of $\Longrightarrow$. For the example given, this grammar formalism would appear to be preferable to either trying to list the infinite set of strings $\{ac, abc, abbc, abbbc, \cdots\}$, or using the informal description $\{w \mid w$ is an a followed by any number of b's followed by a $c\}$. For one thing, it makes feasible the computational task of *parsing* a string to determine whether it is in the language specified by the grammar. A useful byproduct of parsing is the production of a *parse tree* reflecting the grammar rules applied and giving a kind of structural description of grammatical features in the input string—exactly the kind of output that is desired in describing certain biological sequence data.

CFGs have proven to be very useful in the field of compiler construction, where, in the form of BNF (Backus-Naur Form) descriptions, they are used to specify programming languages and their parsers. An even more interesting application of computational linguistics, however, is in understanding natural language—a complex problem that has stimulated a large body of research. Although straightforward CFGs can be written that cover many aspects of natural language syntax, natural languages in their full generality are now thought to require greater than context-free power [8].

Regular expressions also specify languages. However, the set of *regular* languages is strictly a subset of the CFLs, for no regular expression can specify certain *self-embedding* structures such as palindromes; computationally, these require a stack to store information about *dependencies* between distant elements of the string. In fact, even the CFLs are a strict subset of the *context-sensitive* languages (CSLs), described by grammars that have more than one symbol on the LHS of rules. While CFLs are restricted to describing *nested* dependencies, CSLs can specify *crossing* dependencies, such as those found in *copy languages,* which contain duplicated strings of arbitrary extent. These language classes all take their place on the *Chomsky hierarchy* of languages, which categorizes the linguistic complexity of any given language, and which serves as the basis for analysis of the decidability and/or tractability of recognizing strings of any language with general-purpose parsers. $O(n)$ parsers are easily designed for regular and *determinsitic* CFLs—those that can be recognized without the need to backtrack on the input string—and $O(n^3)$ parsers exist for *any* CFL. Certain well-defined characteristics of CFLs may permit more efficient general-purpose parsing, and for any *particular,* narrowly-defined language special-purpose linear-time recognizers can often be designed. Languages beyond context-free are increasingly more difficult to recognize by general-purpose parsers, and much effort has gone into defining language classes "slightly greater" than context-free that are adequate to a particular domain (such as natural language) yet can be parsed efficiently.

The mathematical discipline of formal language theory also provides many tools for evaluating properties of grammars and languages such as their *ambiguity*, referring to strings that may be derived via multiple distinct parses. A simple example of this from natural language would be the sentence *I was given the paper by Watson and Crick*, which with different syntactic parses could either suggest that someone gave me their famous paper, or that those famous persons gave me some paper. Much of the field of Natural Language Processing is concerned with reducing the syntactic ambiguity of sentences by incorporating knowledge of semantics, etc., into the analysis.

We suggested in 1988 [9] that nucleic acids were beyond regular and at least context-free, based on the phenomenon of secondary structure: the stem portion of a stem-and-loop structure entails *nested dependencies* between base-paired residues, which are easily specified by a self-embedding CFG, but which in the general case are beyond the capabilities of any formal regular expression. It was also suggested [9, 10] that DNA may be beyond context-free as well, due to the phenomenon of direct repeats, which constitute a *copy language* with crossing dependencies that cannot be described with essentially stack-based context-free formalisms. Subsequent work [11] formalized these conjectures.

Formal discussion of the linguistic status of DNA, like that of natural language, may be based on empirical phenomenology, but in the case of nucleic acids may also rest on the actual physical structure of the molecules, and in particular the ability to form secondary structure. We have offered formal proofs that idealized representations of such structure are indeed non-regular [11]. Beyond this, however, the simple existence of direct repeats is somewhat unsatisfying as evidence for the purely *formal* status of DNA, since direct repeats can be found even in regular languages (and in fact particular repeats are *required* to exist in both infinite regular languages and CFLs, by the so-called *pumping lemmas* [4]); it is only when direct repeats with no particular bounds on their extent can be shown to be *necessary* in a language that it can be said to be greater than context-free on that account. So as not to depend entirely upon *ad hoc* phenomenology, we sought examples which, like that of inverted repeats, could be grounded in actual physical structures and processes arising in the molecules themselves. A series of such arguments were presented in [11], based upon (1) the potential for circularization of DNA with terminal direct repeats, (2) unequal crossing-over in multiple tandem repeats, and, most importantly, (3) the existence of pseudoknots in structural RNA, which entail crossing dependencies between stems within each other's loops. It is interesting that dealing with pseudoknots has required major reimplementations of some RNA structure prediction programs [1]; the relative difficulty of this can in part be explained by the transition to greater-than-context-free recognition which can thus no longer be strictly stack-based. This points again to the utility of a solid formal linguistic characterization in the design of recognition algorithms in any given domain.

A number of additional formal results were given in [11], dealing with other linguistic attributes, again based on a somewhat idealized model of the structural characteristics of nucleic acids. Inverted repeats, for example, were shown to be *nondeterministic* languages, and the branching or *recursive* nature of secondary structures implies their language is *non-linear* (in this context, meaning that any grammar describing them must have a rule with more than one nonterminal on its right hand side). These results rule out the use of certain $O(n^2)$ simplifications of general-purpose context-free parsers (which are otherwise $O(n^3)$).

The *ambiguity* of general secondary structural grammars (that is, their ability to produce more than one essentially distinct parse for the same primary sequence) was also explored in [11], and it was shown that this grammatical ambiguity reflects alternative secondary structures in a biologically relevant way. (Since that time, we have proven that, while the most general language of ideal orthodox secondary structure is actually deterministic and thus unambiguous, certain biologically plausible secondary structure sublanguages are *inherently ambiguous*, i.e. impossible to describe by any unambiguous grammar). An understanding of the nature and degree of ambiguity of languages in a domain is important, for example in implementing deterministic speed-ups to parsers.

Given that biological sequences are beyond context-free, it is of interest to carefully circumscribe their exact boundaries. We suggested in [10], and demonstrated in [11], that the language encompassing all of the phenomena described above in nucleic acid structure lies not only in the CSLs, but within a restricted subset known as the *indexed languages.* (It is interesting that the indexed languages have also been claimed to suffice for natural languages [2].) A subset of the indexed languages with a very perspicuous grammar formalism particularly well-suited to nucleic acids, known as *string variable grammar,* was developed by us [10, 11]; examples are given in the next section.

We have also explored *closure* properties of the Chomsky hierarchy under biological operations—that is, whether language classes of interest, after undergoing certain biological processes, can be expected to remain at the same level in the Chomsky hierarchy or not. It was formally demonstrated in [11] that regular languages and CFLs are closed under double-stranded replication and under simple recombinational events such as scission and ligation. Deterministic languages, however, are not closed under these operations, so that, for example, certain features can be recognized more efficiently on one strand of DNA than on the other, or in other than a leftmost fashion, suggesting the use of so-called *island parsing* strategies. With regard to *evolutionary* operations such as duplication, inversion, and transposition, it was shown that CFLs are *not* closed, suggesting that genomic rearrangements on an evolutionary scale may be responsible for increasing the mathematical complexity of the genetic language.

One of the more remarkable aspects of grammar-based descriptions of folded structure is the observation that derivation trees from the grammars physically resemble the actual secondary structures as they are usually portrayed. This has led to several machine learning approaches to prediction of secondary structure based on *stochastic grammars* [3, 7], and has caused us to examine new formalisms that stress the structural aspects of derivation trees for grammars that are beyond context-free. Among these are *tree-adjoining grammars* and variations upon them [5, 6]. Recently, the author has developed a new formalism that directly addresses the problem of representing relationships *between* strings in a language, rather than just *within* the same string. A *cut grammar* has a new symbol that allows a derived string to be cut at multiple places, so that the language derived is actually a *set* of strings all related by having come from the same derivation [12]. This allows us to express the results of hybridization of oligonucleotides, for example.

References

[1] 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.

[2] G. Gazdar. Applicability of indexed grammars to natural languages. Technical Report CSLI-85-34, Center for the Study of Language and Information, Stanford, 1985.

[3] L. Grate, M. Herbster, R. Hughey, I.S. Mian, H. Noller, and D. Haussler. RNA modeling using Gibbs sampling and stochastic context free grammars. In *Proc. of Second Int. Conf. on Intelligent Systems for Molecular Biology*, Menlo Park, CA, August 1994. AAAI/MIT Press.

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

[5] A. K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87–114. John Benjamin, Amsterdam, 1987.

[6] A.K. Joshi, K. Vijay-Shanker, and D. Weir. The convergence of mildly context-sensitive grammar formalisms. In P. Sells, S.M. Shieber, and T. Wasow, editors, *Foundational Issues in Natural Language Processing*, pages 31–81. MIT Press, Cambridge, MA, 1991.

[7] Y. Sakakibara, M. Brown, I.S. Mian, R. Underwood, and D. Haussler. Stochastic context-free grammars for modeling RNA. In *Proceedings of the Hawaii International Conference on System Sciences*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[8] S.M. Schieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

[9] D. B. Searls. Representing genetic information with formal grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 386–391. American Association for Artificial Intelligence, 1988.

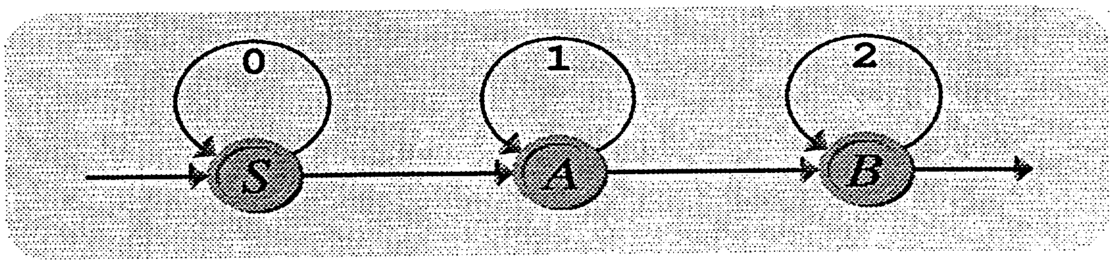
[10] D. B. Searls. Investigating the linguistics of DNA with definite clause grammars. In E. Lusk and R. Overbeek, editors, *Logic Programming: Proceedings of the North American Conference*, pages 189–208. MIT Press, 1989.

[11] D. B. Searls. The computational linguistics of biological sequences. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, chapter 2, pages 47–120. AAAI Press, 1993.

[12] D.B. Searls. Formal grammars for intermolecular structure. *First International IEEE Symposium on Intelligence in Neural and Biological Systems*, pages 30–37, 1995.

Regular Languages

- regular languages are those generated by simple finite state automata, and (equivalently) by grammars whose rewrite rules or *productions* have only single *nonterminals* at the ends of their right hand sides, e.g. for $\{\, 0^i 1^j 2^k \mid i, j, k \geq 0\}$:

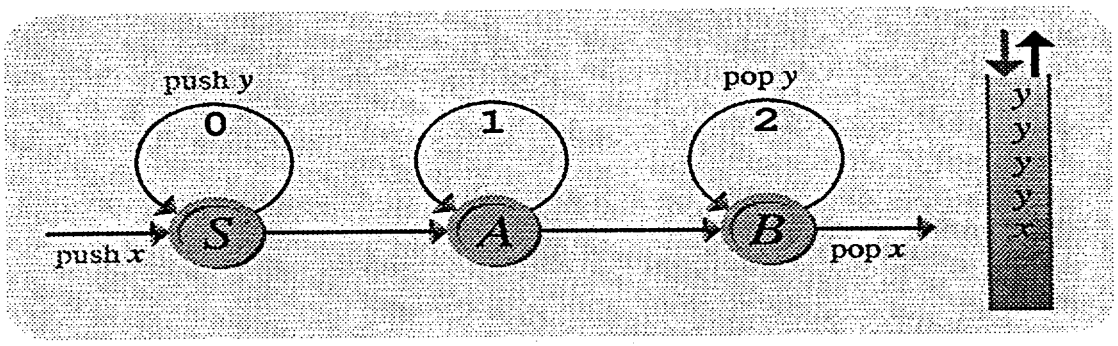


$S \longrightarrow 0S \mid A \qquad A \longrightarrow 1A \mid B \qquad B \longrightarrow 2B \mid \varepsilon$

$\underline{S} \Rightarrow 0\underline{S} \Rightarrow 0\underline{A} \Rightarrow 01\underline{A} \Rightarrow 011\underline{A} \Rightarrow 0111\underline{A} \Rightarrow$
$\Rightarrow 0111\underline{B} \Rightarrow 01112\underline{B} \Rightarrow 011122\underline{B} \Rightarrow 011122$

Context-Free Languages

- context-free languages are modeled by pushdown automata, which have memory in the form of a *stack*, and by grammars with no limitations on their right hand sides, e.g. for $\{\, 0^i 1^j 2^i \mid i, j \geq 0\}$:

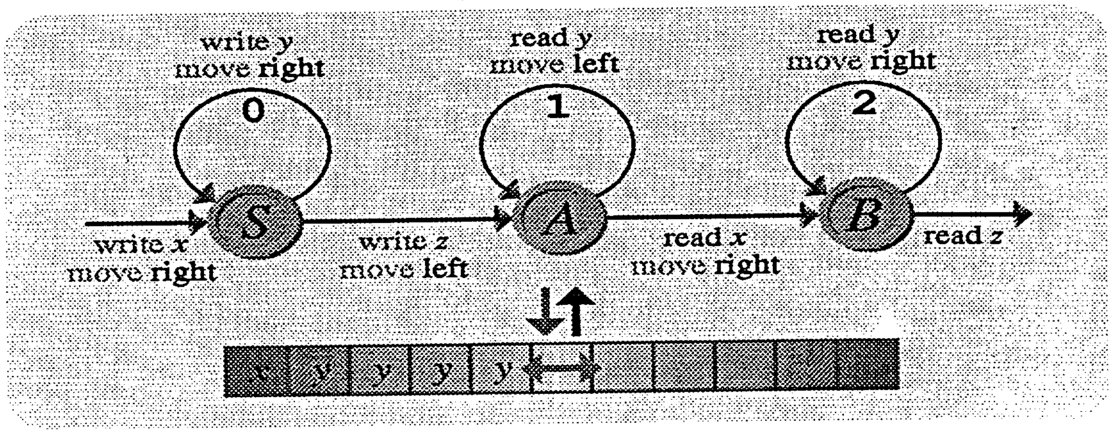


$S \longrightarrow 0S2 \mid A \qquad A \longrightarrow 1A \mid \varepsilon$

$\underline{S} \Rightarrow 0\underline{S}2 \Rightarrow 00\underline{S}22 \Rightarrow 000\underline{S}222 \Rightarrow 000\underline{A}222 \Rightarrow$
$\Rightarrow 0001\underline{A}222 \Rightarrow 00011\underline{A}222 \Rightarrow 00011222$

Beyond Context-Free

- $\{ 0^i 1^i 2^i \mid i, j, k \geq 0 \}$ requires even greater power:



$S \longrightarrow 0SA2 \mid 012 \qquad 1A \longrightarrow 11 \qquad 2A \longrightarrow A2$

$\underline{S} \Rightarrow 0\underline{S}A2 \Rightarrow 00\underline{S}A2A2 \Rightarrow 0001\underline{2A}2A2 \Rightarrow$
$\Rightarrow 000\underline{1A}22A2 \Rightarrow 000112\underline{2A}2 \Rightarrow 00011\underline{2A}22 \Rightarrow$
$\Rightarrow 0001\underline{1A}222 \Rightarrow 000111222$

The Chomsky Hierarchy

- *recursively enumerable* languages use an infinite tape for memory, i.e. a Turing machine, and are equivalent to grammars with any number of additional symbols on their left hand sides
- *context sensitive* languages require a tape bounded in the size of the input, and left hand sides cannot be longer than right hand sides
- the *Chomsky hierarchy*, $RL \subset CFL \subset CSL \subset RE$, establishes the subsumption relationships
- in ascending the Chomsky hierarchy, language classes become less tractable in terms of recognition *(parsing)*, decidability and closure properties, etc.
- natural language is thought to be "slightly greater" than context free

Reverse Complementarity (I)

- we will uniformly adopt the alphabet of DNA:

$$\Sigma_{\text{DNA}} = \{\text{g}, \text{c}, \text{a}, \text{t}\} \tag{1}$$

- the following function indicates bases that are able to physically and informationally *base-pair* between strands of double-helical DNA:

$$\overline{\text{g}} = \text{c}, \quad \overline{\text{c}} = \text{g}, \quad \overline{\text{a}} = \text{t}, \quad \text{and} \quad \overline{\text{t}} = \text{a} \tag{2}$$

- this operation can be extended over strings and constitutes a *homomorphism,* since we can say that

$$\overline{u} \cdot \overline{v} = \overline{(uv)} \text{ for } u, v \in \Sigma_{\text{DNA}}^*, \quad \text{and} \quad \overline{\varepsilon} = \varepsilon \tag{3}$$

- we will abbreviate (3) as $\overline{uv}$; this homomorphism and string reversal have the following properties:

$$\overline{(\overline{w})} = w, \quad (w^R)^R = w, \quad \text{and} \quad \overline{(w^R)} = (\overline{w})^R \tag{4}$$

Reverse Complementarity (II)

- the composition of base complementarity and reversal, written $\overline{w}^R$, is the "opposite strand" of a string w of DNA; it is *not* a homomorphism, since

$$\overline{u}^R \cdot \overline{v}^R \neq \overline{(uv)}^R = \overline{v}^R \cdot \overline{u}^R \quad \text{where } u \neq v \tag{5}$$

- rather, it is a group-theoretic *involution*; this allows DNA to be *replicated* from opposite strands:

$$\overline{(\overline{w}^R)}^R = \overline{((\overline{w}^R)^R)} = \overline{(\overline{w})} = w \tag{6}$$

- consider strings identical to their opposite strands:

$$L_o = \{w \in \Sigma_{\text{DNA}}^* \mid w = \overline{w}^R\} \tag{7}$$

- dividing any such w into equal halves, we see that

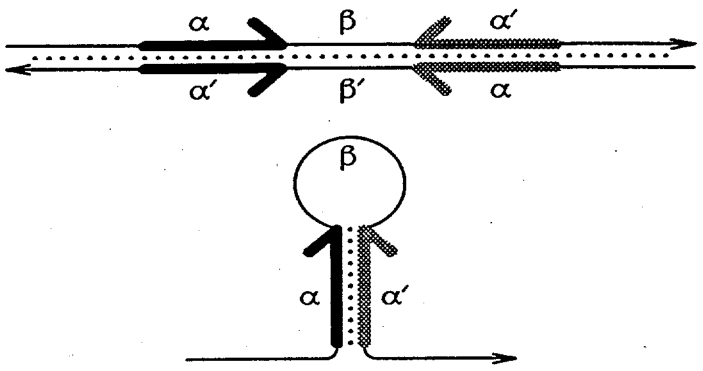
$$w = uv = \overline{w}^R = \overline{v}^R\overline{u}^R = u\overline{u}^R \quad \text{where } |u| = |v| \tag{8}$$

so that L_o is in fact the language

$$L_o = \{u\overline{u}^R \mid u \in \Sigma_{\text{DNA}}^*\} \tag{9}$$

Inverted Repeats

- *inverted repeats* are common in nucleic acids:



- we could specify these with the following grammar:

$$S \longrightarrow bS\bar{b} \mid A \qquad A \longrightarrow bA \mid \varepsilon \qquad \text{where } b \in \Sigma_{\text{DNA}} \quad (10)$$

- however, the A rule (for the loop) can specify any string, so the resulting language is Σ^*_{DNA} and trivially regular; therefore, we idealize to *hairpins*:

$$S \longrightarrow bS\bar{b} \mid \varepsilon \qquad (11)$$

DNA is Not Regular

- let an *ideal* string be one with equal numbers of each base type b and its complement $\bar{b}$ (i.e. all bases are potentially paired); the self-embedding context-free grammar $S \longrightarrow bS\bar{b} \mid \varepsilon$ of ideal inverted repeats in fact yields L_o, and is clearly not regular
- one way to model non-ideal inverted repeats would be to require a minimum length p for the stem and a maximum length q for the loop:

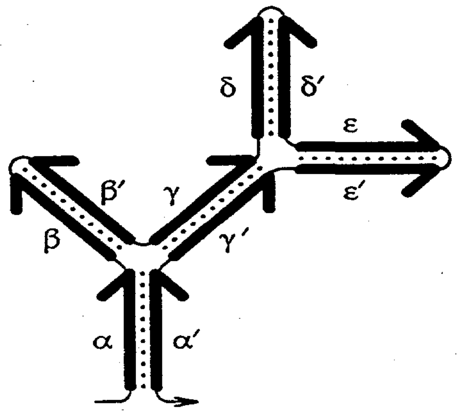
$$L_n = \{uv\bar{u}^R \mid u, v \in \Sigma^*_{\text{DNA}},\ |u| \geq p, \text{ and } |v| \leq q\} \quad (12)$$

- this is non-regular and still context-free, even "biological", but may not find the longest stem; for purposes of recognition, the following is more useful:

$$\begin{aligned} S &\longrightarrow bS\bar{b} \mid A \\ A &\longrightarrow b \mid \epsilon \mid bBd \qquad \text{where } b \neq d \\ B &\longrightarrow bB \mid \varepsilon \end{aligned} \qquad (13)$$

DNA Is Not Linear

- nucleic acids form recursive *secondary structure:*



- let an *orthodox* string be ε, or the result of adding an adjacent complementary pair $b\bar{b}$ at any position in an orthodox string; this language is given by

$$S \rightarrow bS\bar{b} \mid SS \mid \varepsilon \tag{14}$$

DNA Is Nondeterministic

- the language L_o of ideal inverted repeats requires guessing about the midpoint, so is nondeterministic
- surprisingly, the more general language L_s of orthodox secondary structure is deterministic; though the grammar given above is nondeterminstic, it is weakly equivalent to the deterministic Griebach normal form grammar

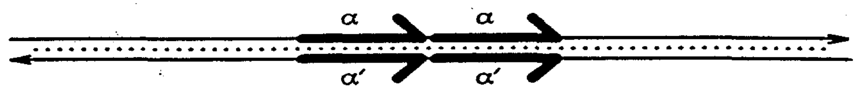
$$\begin{array}{rcll} S & \rightarrow & bS_bS \mid \varepsilon & \text{for each } b \in \Sigma_{\text{DNA}} \\ S_b & \rightarrow & \bar{b} \mid dS_dS_b & \text{for } d \neq \bar{b} \end{array} \tag{15}$$

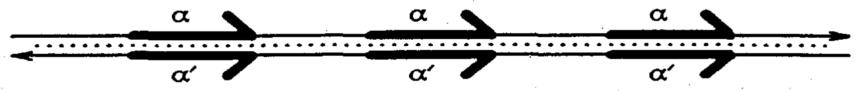
- nevertheless, constraining this very general language of secondary structure to give specific nonlinear languages may reintroduce nondeterminism, for example the *dumbbell* language of adjacent stems, $L_d = \{u\bar{u}^R v\bar{v}^R \mid u, v \in \Sigma^*_{\text{DNA}}\}$, specified by

$$S \rightarrow AA \qquad A \rightarrow bA\bar{b} \mid \varepsilon \tag{16}$$

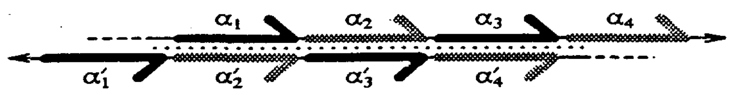
DNA Is Not Context-Free (I)

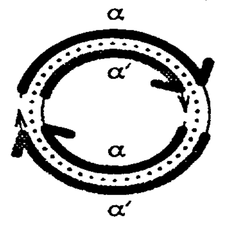
- tandem and direct repeats are frequent in DNA:





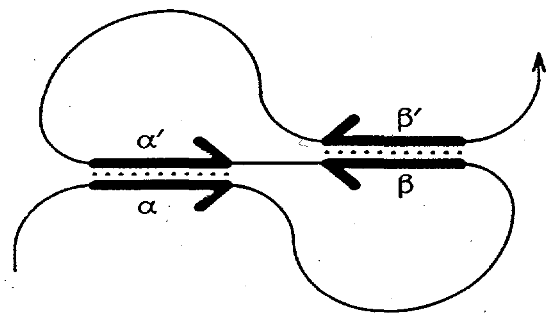
- these are *copy languages*, e.g. $L_c = \{ww \mid w \in \Sigma^*_{\text{DNA}}\}$, which are beyond context-free; structural correlates may lead to alternative hybridization, unequal crossing over, even circularization:





DNA Is Not Context-Free (II)

- also, consider RNA *pseudoknots*:



- an ideal language of pseudoknots would be

$$L_k = \{uv\overline{u}^R\overline{v}^R \mid u, v \in \Sigma^*_{\text{DNA}}\} \tag{17}$$

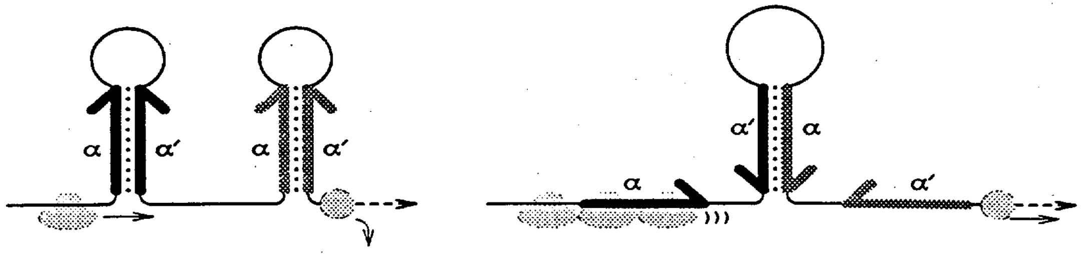
which is not context-free, since

$$\begin{aligned} L'_k &= L_k \cap g^+a^+c^+t^+ \\ &= \{\mathrm{g}^i\mathrm{a}^j\mathrm{c}^i\mathrm{t}^j \mid i, j \geq 1\} \end{aligned} \tag{18}$$

is not only non-orthodox, but homomorphic to a well-known non-context-free language

DNA Is Not Context-Free (III)

- *attenuators* are binary switches that control certain bacterial genes using *alternative secondary structure*



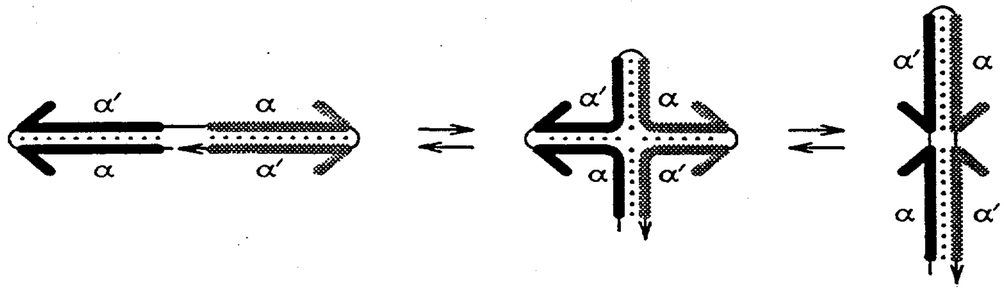
- these occur in forms whose formal expressions are beyond context-free, since they contain copies:

$$L_{a1} = \{w\overline{w}^R w \mid w \in \Sigma^*_{\text{DNA}}\} \tag{19}$$
$$L_{a2} = \{w\overline{w}^R w\overline{w}^R \mid w \in \Sigma^*_{\text{DNA}}\} \tag{20}$$

- L_{a2} is a subset of both the hairpin language L_o and the dumbbell language L_d, and in fact is the intersection of either with the copy language

DNA Is Inherently Ambiguous (I)

- in fact, any grammar for L_{a2} must give rise to multiple leftmost derivations (as does the orthodox):



$$\begin{aligned}
S &\Rightarrow SS \Rightarrow \mathrm{g}S\mathrm{c}S \Rightarrow \mathrm{ga}S\mathrm{tc}S \Rightarrow \mathrm{gatc}S \\
&\Rightarrow \mathrm{gatcg}S\mathrm{c} \Rightarrow \mathrm{gatcga}S\mathrm{tc} \Rightarrow \mathrm{gatcgatc} \\
S &\Rightarrow \mathrm{g}S\mathrm{c} \Rightarrow \mathrm{g}SS\mathrm{c} \Rightarrow \mathrm{g}SSS\mathrm{c} \Rightarrow \mathrm{ga}S\mathrm{t}SS\mathrm{c} \\
&\Rightarrow \mathrm{gat}SS\mathrm{c} \Rightarrow \mathrm{gatc}S\mathrm{g}S\mathrm{c} \Rightarrow \mathrm{gatcg}S\mathrm{c} \\
&\Rightarrow \mathrm{gatcga}S\mathrm{tc} \Rightarrow \mathrm{gatcgatc} \\
S &\Rightarrow \mathrm{g}S\mathrm{c} \Rightarrow \mathrm{ga}S\mathrm{tc} \Rightarrow \mathrm{gat}S\mathrm{atc} \\
&\Rightarrow \mathrm{gatc}S\mathrm{gatc} \Rightarrow \mathrm{gatcgatc}
\end{aligned} \tag{21}$$

DNA Is Inherently Ambiguous (II)

- alternative structures like those above are obviously not captured by the deterministic Griebach normal form grammar for L_s, which returns only one structure for any input
- on the other hand, the straightforward ambiguous grammar for L_s also generates more than one leftmost derivation for the same secondary structure, i.e. it is also *structurally ambiguous*
- we can design a *structurally unambiguous* grammar, that generates exactly one leftmost derivation per secondary structure, though the degree of ambiguity is still exponential in the size of the input:

$$\begin{aligned} S &\rightarrow A \mid \varepsilon \\ A &\rightarrow bA\bar{b} \mid AB \mid b\bar{b} \\ B &\rightarrow bA\bar{b} \mid b\bar{b} \end{aligned} \tag{22}$$

Closure under Replication

- consider the operation of replication, for $L \subseteq \Sigma^*_{\text{DNA}}$:

$$\text{REP}(L) = \{w, \overline{w}^R \mid w \in L\} = L \cup \overline{L}^R \tag{23}$$

- the languages in the Chomsky hierarchy are all closed under the operations of homomorphism, string reversal, and union, and so also under replication; in fact, we observe a fixpoint:

$$\text{REP}^*(L) = \text{REP}(L) \tag{24}$$

- but, deterministic context-free languages, e.g.

$$L_D = \{\text{g}^i\text{a}^j\text{t}^k\text{c}^* \mid i = j + k\} \tag{25}$$

are *not* closed under replication, since

$$\text{REP}(L_D) = \{\text{g}^p\text{a}^q\text{t}^r\text{c}^s \mid p = q + r \ \text{ or } \ s = q + r\} \tag{26}$$

is not only nondeterministic but inherently ambiguous, necessarily having multiple leftmost derivations whenever $p = q + r = s$

Closure under Recombination

- the classifications of the Chomsky hierarchy are also closed under ligation (and its closure)
$$\text{LIG}(L) = \{xy \mid x, y \in L\} = L \cdot L \tag{27}$$
since DNA can only ligate head-to-tail, and this holds true even in populations of double-stranded DNA:
$$\begin{aligned}\text{LIG}(\text{REP}(L)) &= \text{LIG}(L \cup \overline{L}^R) \\ &= (L \cdot L) \cup (L \cdot \overline{L}^R) \cup (\overline{L}^R \cdot L) \cup (\overline{L}^R \cdot \overline{L}^R)\end{aligned} \tag{28}$$
- the same is true of scission (and its closure)
$$\begin{aligned}\text{CUT}(L) &= \{x, y \mid xy \in L\} = \text{PRE}(L) \cup \text{SUF}(L) \\ \text{CUT}^*(L) &= \{u \mid xuy \in L\} = \text{PRE}(\text{SUF}(L))\end{aligned} \tag{29}$$
- although we cannot directly model ligation which circularizes strings, we can model their scission:
$$\text{CUT}(\text{LIG}^{\bigcirc}(L)) = \{vu \mid uv \in L\} = \text{CYC}(L) \tag{30}$$

Closure under Evolution

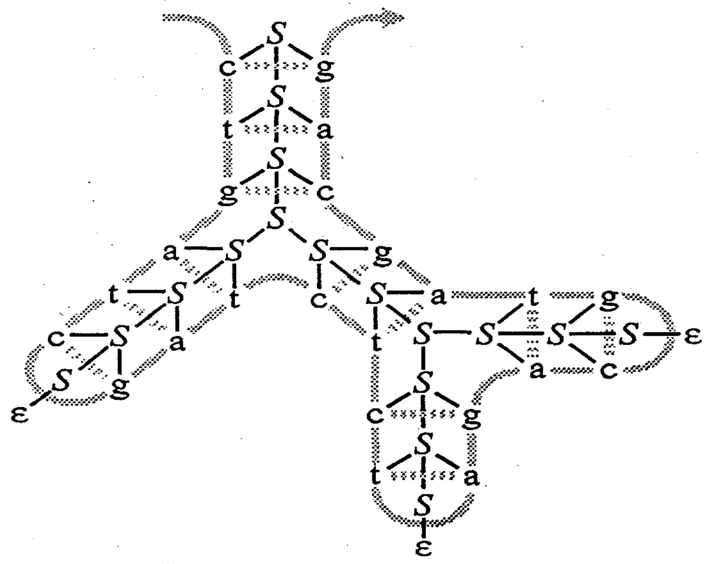
- evolutionary rearrangements can also be modelled:
$$\begin{aligned}\text{DUP}(L) &= \{xuuy \mid xuy \in L\} \\ \text{INV}(L) &= \{x\overline{u}^R y \mid xuy \in L\} \\ \text{XPOS}(L) &= \{xvuy \mid xuvy \in L\} \\ \text{DEL}(L) &= \{xy \mid xuy \in L\}\end{aligned} \tag{31}$$
where $x, y, u, v \in \Sigma^*_{\text{DNA}}$ and $L \subseteq \Sigma^*_{\text{DNA}}$
- regular or context-free languages could not be closed under duplication, since this creates copy languages; neither are they closed under inversion (which makes copy languages from inverted repeats) or transposition (which makes pseudoknots from them)
- only under deletion are the lower levels of the Chomsky hierarchy preserved – thus, evolution may tend towards increasing linguistic complexity

Closure under Expression

- during gene expression, transcription, processing, and translation may take place at different times and/or in different compartments of the cell
- thus, the signals relevant to the DNA, various forms of RNA, and protein, are all projected back to the DNA, and to the extent these can or should be viewed as separate languages, the DNA must be seen as the *intersection* of those languages
- this is significant since (for example) the context-free languages are not closed under intersection
- there is evidence that secondary structure may play a role in expression (e.g. regulating alternative splicing), and in fact it may interfere with ribosome binding – and context free languages are not closed under complementation

Parse Trees and Structure

- secondary structure grammar derivations not only capture base-pairing dependencies in rules, but the resulting trees resemble the overall structure



DNA As An Indexed Language

- the language of all ideal strings can be shown to be context-sensitive; however, the phenomena in DNA may be subsumed by the *indexed* languages, which lie between context-free and -sensitive
- indexed grammars can be thought of as having stacks attached to nonterminals, which are passed along to each nonterminal arising in a derivation, e.g. for tandem repeats

$$S \rightarrow bS^{b} \mid A \qquad A^{b} \rightarrow Ab \qquad A \rightarrow \varepsilon \tag{32}$$

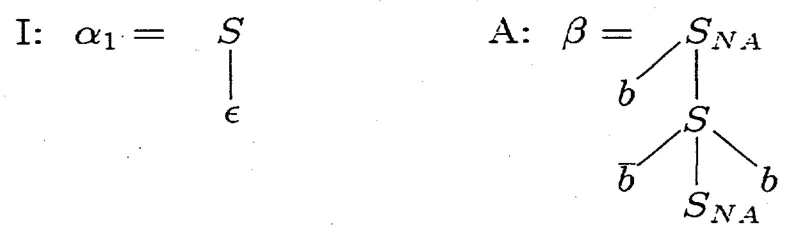
gives rise to

$$\begin{aligned} S &\Rightarrow \mathrm{g}S^{\mathrm{g}} \Rightarrow \mathrm{gc}S^{\mathrm{cg}} \Rightarrow \mathrm{gca}S^{\mathrm{acg}} \Rightarrow \mathrm{gca}A^{\mathrm{acg}} \Rightarrow \\ &\mathrm{gca}A^{\mathrm{cg}}\mathrm{a} \Rightarrow \mathrm{gca}A^{\mathrm{g}}\mathrm{ca} \Rightarrow \mathrm{gca}A\mathrm{gca} \Rightarrow \mathrm{gcagca} \end{aligned} \tag{33}$$

- similar grammars suffice for inverted repeats, pseudoknots, etc. of seemingly arbitrary complexity

DNA as a TAG Language

- certain biologically-relevant languages that are beyond context-free can be captured as a more tractable *tree-adjoining grammar*, e.g. the attenuator language $L_{a1} = \{w\overline{w}^{R}w \mid w \in \Sigma^{*}_{\text{DNA}}\}$:

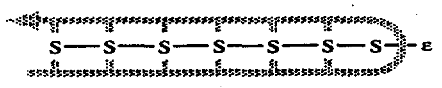


(34)

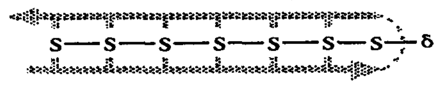
- this grammar captures base-pairing *and* alternative base-pairing in constituent structure
- however, TAGs do not appear to handle L_{a2} or the pseudoknot language $L_{k} = \{uv\overline{u}^{R}\overline{v}^{R} \mid u, v \in \Sigma^{*}_{\text{DNA}}\}$ (though extensions such as multicomponent TAGS might suffice)

Cut Grammars

- a cut grammar is an ordinary grammar with a new symbol, δ, at which derived strings are simply cut



$$S \rightarrow bS\overline{b} \mid \varepsilon \tag{35}$$



$$S \rightarrow bS\overline{b} \mid \delta \tag{36}$$

- this will allow us to use grammars to describe *inter*molecular as well as intramolecular interactions
- given a string $u = u_1\delta u_2\delta \cdots \delta u_n$ where $u_i \in \Sigma^*$ for $1 \leq i \leq n$ (and $\delta \notin \Sigma$), we define a *cut function*

$$\hat{u} \stackrel{\mathrm{def}}{=} \{u_1, u_2, \cdots, u_n\} \tag{37}$$

and an *uncut function*

$$\tilde{u} \stackrel{\mathrm{def}}{=} u_1 u_2 \cdots u_n \tag{38}$$

Cut Languages

- for a given cut grammar G with start symbol S, we define the *cut language*

$$\hat{L}(G) \stackrel{\mathrm{def}}{=} \{\, \hat{u} \in 2^{\Sigma^*} \mid S \Longrightarrow^* u\} \tag{39}$$

(a set of sets), and the *uncut language*

$$\tilde{L}(G) \stackrel{\mathrm{def}}{=} \{\, \tilde{u} \in \Sigma^* \mid S \Longrightarrow^* u\} \tag{40}$$

- we will also consider the *cut language union*

$$\cup \hat{L}(G) = \{\, u \in \Sigma^* \mid S \Longrightarrow^* v \text{ and } u \in \hat{v}\} \tag{41}$$

- for the double-strand grammar $G_o : S \rightarrow bS\overline{b} \mid \delta$,

$$\hat{L}(G_o) = \{\, \{u, \overline{u}^R\} \mid u \in \Sigma^*_{\mathrm{DNA}}\,\} \tag{42}$$

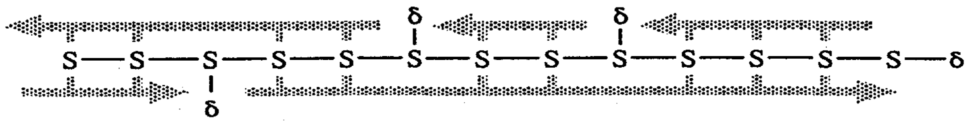
which is related to the stem language by

$$L_o = \tilde{L}(G_o) = \{\, u \mid \{u\} \in \hat{L}(G_o)\} \tag{43}$$

Nick Languages

- *nicked* double-stranded DNA can be modelled by

$$S \longrightarrow bS\overline{b} \mid \delta S \mid S\delta \mid \delta \qquad (44)$$



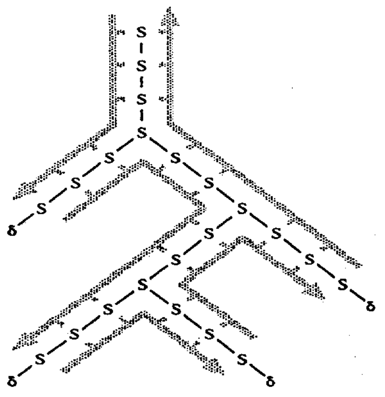
- we can also require a minimum "overhang" to create what biologists call "sticky ends":

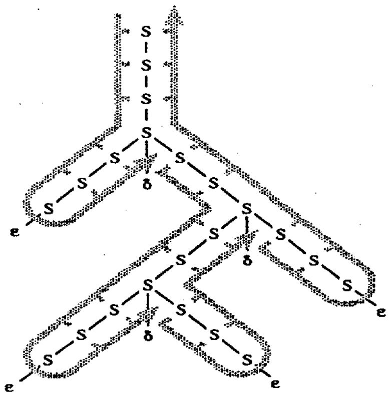
$$S \longrightarrow bS\overline{b} \mid w\delta S\overline{w}^R \mid wS\delta\overline{w}^R \mid \delta \qquad (45)$$

  for each $w \in \Sigma^n_{\text{DNA}}$ where n is the desired length (or for particular w's to model restriction enzyme sites)

- thus, cut grammars can be used to formally describe *hybridization* of populations of strings, e.g. cut language elements as sets of hybridizable "oligos"

Nonlinear Cut Languages





- by analogy with orthodox structure, we can model
  - end-cut hybridization: $S \longrightarrow bS\overline{b} \mid SS \mid \delta$
  - fork-cut hybridization: $S \longrightarrow bS\overline{b} \mid S\delta S \mid \epsilon$
  - generalized hybridization networks:

$$S \longrightarrow bS\overline{b} \mid SS \mid S\delta \mid \delta S \mid \epsilon \qquad (46)$$

Circular Cut Languages

- using the start symbol to leave one end of the double-stranded molecule open, and a δ to cut open the other end, seems arbitrary given the symmetry
- to "close off" the start of a derivation tree, we can define a *circular cut function*

$$\mathring{u} \overset{\text{def}}{=} \{u_n u_1, u_2, u_3, \cdots, u_{n-1}\} \tag{47}$$

  which is only defined when at least one δ is derived
- then, for $G_o : S \longrightarrow bS\bar{b} \mid \delta$, we have ordinary stems

$$\cup\, \mathring{L}(G_o) = \tilde{L}(G_o) = L_o \tag{48}$$

  which is to say, the set of stems open at the start is the same as the set open at the terminus
- for any G we can form a G' by adding a new start symbol S' and rule $S' \rightarrow S\delta$, to get $\mathring{L}(G') = \hat{L}(G)$

Ligated Languages

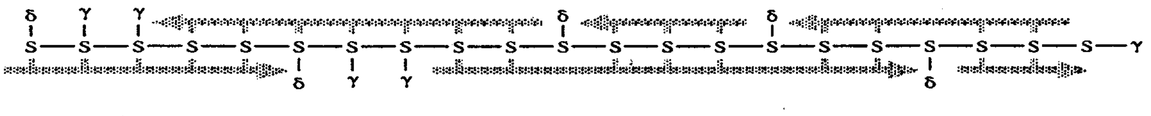
- we may wish to distinguish ligateable cuts from non-ligateable ones, e.g. nicks vs. gaps or ends
- a *ligation grammar* is a cut grammar with an additional new symbol γ, where for any $u = u_1\gamma u_2\gamma \cdots \gamma u_n$ with $u_i \in (\Sigma \cup \{\delta\})^*$ for each $1 \leq i \leq n$, we define the *ligate function*

$$\breve{u} = \{\tilde{u}_1, \tilde{u}_2, \cdots, \tilde{u}_n\} \tag{49}$$

- given a ligation grammar G with start symbol S we define the *ligated language*

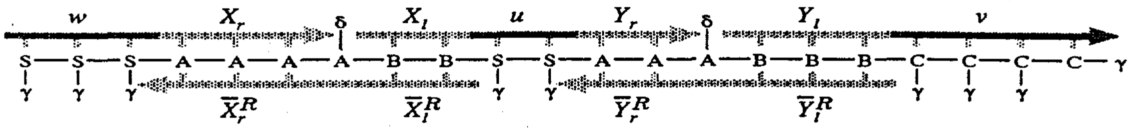
$$\breve{L}(G) = \{\ \breve{u} \in 2^{\Sigma^*} \mid S \Longrightarrow^* u\} \tag{50}$$

  while the cuts act on both δ's and γ's, as before



Regular Hybridization

- given a right-linear grammar R, i.e. one with rules of the form $A \longrightarrow wB$ and $A \longrightarrow w$, we can create *oligonucleotides* encoding the rules of the grammar, together with additional *splint* oligos, so as to hybridize to structures like



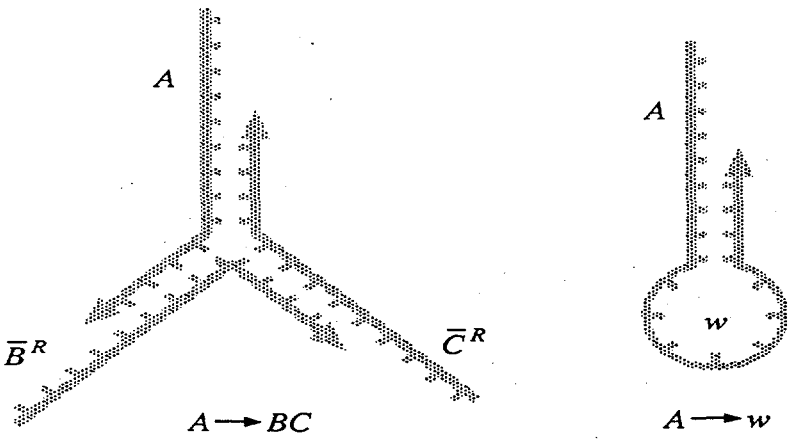
which are described by a grammar G_h:

$$\begin{aligned} S &\longrightarrow \gamma S b \mid A b \\ A &\longrightarrow b A \overline{b} \mid B \delta \\ B &\longrightarrow b B \overline{b} \mid b S \overline{b} \mid b C \overline{b} \\ C &\longrightarrow \gamma C b \mid \gamma b \end{aligned} \tag{51}$$

- arbitrary derivations from R can be produced by "complete" hybridization

Context-Free Hybridization

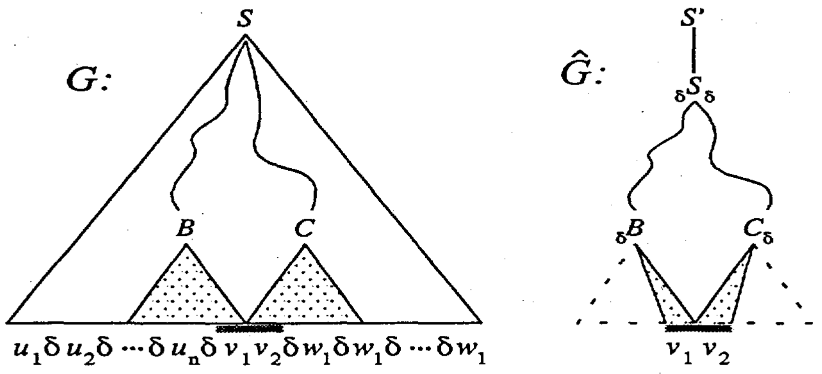
- by allowing branched hybridization, we can similarly model arbitrary context-free grammars (in Chomsky normal form) with oligos like the following:



- could non-context-free secondary structures derive arbitrary non-context-free languages?

Cut Language Closure

- given a context-free cut grammar G, we can
  - change each δ to ϵ, and for this ordinary $\tilde{G}$ clearly $L(\tilde{G}) = \tilde{L}(G)$, so $\tilde{L}(G)$ is context-free
  - create an ordinary $\hat{G}$ (by a construction due to Tilman Becker) which "chooses" each pair of adjacent δ's and generates only the intervening substring, so that $\cup\hat{L}(G) = L(\hat{G})$ is context free


G:
S
B
C
$u_1\delta u_2\delta \cdots \delta u_n\delta v_1 v_2\delta w_1\delta w_1\delta \cdots \delta w_1$
$\hat{G}$:
S'
$_\delta S_\delta$
$_\delta B$
C_δ
$v_1\, v_2$

Cut Language Recognition

- the membership problem for cut languages: given a cut grammar G and a set $V \subseteq \Sigma^*$, is $V \in \hat{L}(G)$?
- this problem is NP-hard, by reduction from Directed Hamiltonian Path (due to Michael Niv): Given a graph (V, E), where $|V| = n$, the start vertex is v_1, and the end vertex is v_n, if we define a cut grammar with alphabet $\Sigma = V$, nonterminals $\{A^i_j \mid 1 \le i, j \le n\} \cup \{S\}$, and rules

$$\begin{array}{lll} S & \longrightarrow & v_1 A^1_1 \\ A^i_j & \longrightarrow & \delta v_k A^{i+1}_k \quad \text{for all } (j,k) \in E,\ 1 \le i < n \\ A^n_n & \longrightarrow & \epsilon \end{array} \tag{52}$$

  then $V \in \hat{L}(G)$ *iff* there is a path of length n that passes through every vertex in V just once (DHP)
- ironically, Adleman "solved" DHP by hybridization

3 Implementation

Linguistic methodology is not limited to computer languages or human natural languages, but can be extended to all manner of signals, images, or other data which have underlying structure. This observation has led to the development of the field of Syntactic Pattern Recognition (SPR) [6]. SPR makes use of the tools and techniques of computational linguistics, such as grammars and parsers, to specify and search for patterns in data. Because grammars intrinsically promote the hierarchical abstraction of features, these can be built up to a very high level while maintaining a clear, modular "knowledge base." Moreover, grammars by their nature detect individual features in this higher-level context, which creates a much greater degree of discrimination than isolated searches. SPR benefits from a strong formal foundation, but also incorporates features that extend the expressive power of grammars where necessary for the domain. For example, "noisy" signals can be dealt with by so-called stochastic grammars [6], which incorporate probabilities into grammars in a natural way. SPR, in fact, has been classified as a form of *pattern-directed inference*, and indeed we have found that it provides an excellent framework for the incorporation of heuristics at many levels. SPR has been successfully applied to such problems as general signal processing, handwritten character recognition, and karyotype analysis by the author [17, 19] and many others [6], and our results with SPR and the linguistic analysis of DNA suggest that they are appropriate approaches to the complexities of this domain as well.

For this purpose we use the Prolog programming language, which implements a procedural interpretation of a subset of first-order predicate logic. It uses a particular clausal form that allows programs to be written as databases containing atomic predicates called *facts*, e.g., `protein(hemoglobin)`, and *rules* which are written in the form `protease(X) :- protein(Y), degrades(X,Y)`. This can be read "X is a protease *if* Y is a protein *and* X degrades Y." Prolog's rules and facts, together called *relations*, can be queried to perform inferences by backward-chaining proof, using a mechanism called *resolution*, and the resulting system is able to perform computation as controlled deduction—in fact, a form of theorem proving.

Prolog's history is closely linked with the formalism of Definite Clause Grammars (DCGs), and the notion that grammars can be expressed as rules of a Prolog program [14]. The process of parsing a string then becomes that of proving a theorem given that string as input and the "axioms" of a grammar. In practice, such a grammar would appear as in the code below.

```
s --> [a], x.
x --> [b], x | [c].
```

In Prolog, logical predicates begin with a lower-case letter, and in DCGs these correspond to nonterminals. Terminals are shown as Prolog list elements; lists generally appear within square brackets, with list elements separated by commas (e.g., `[a,b,c]`). The vertical bar in the second rule is an "or" (disjunction).

DCGs actually require a translation step to become Prolog clauses, because Prolog must have a mechanism for manipulating the input string, which it does by maintaining the string in "hidden" parameters of the nonterminals. The first DCG rule above would be translated to the following Prolog rule:

```
s(S0,S) :- S0=[a|S1], x(S1,S).
x(S0,S) :- S0=[b|S1], x(S1,S) ; S0=[c|S].
```

(Note that variables in Prolog begin with upper-case letters). When nonterminals are translated, they have two variable parameters added—sometimes called *difference lists*—corresponding to the lists that will be passed in and then back out, i.e. the input string and what is left of it after the nonterminal consumes some initial string from it. Terminals are translated such that the specified alphabetic elements are "consumed" from the front of the input list. The difference lists are arranged so that the span of the LHS nonterminal is that of the entire RHS. Thus, actual top-level calls to `s` would succeed in forms such as `s([a,c],[])` or `s([a,b,b,c],[])` (with the empty list being the necessary remainder after the parse succeeds); in our implementation, a double-arrow infix operator is used to express such queries, following the formal notation, e.g. `s ==> "abbc"`. Note also that a useful alternative notation for lists is as strings within double quotes (whose elements actually correspond to ASCII character codes); we will use this for DNA, e.g., `"gattac"`.

DCGs actually have expressive power far beyond context-free, by virtue of the fact that Prolog code can be freely embedded (within curly braces); in addition, parameters may be attached to nonterminals, and terminals may appear following the left hand side nonterminal. We have used DCGs to develop a syntactic pattern recognition system for DNA sequences, known as GENLANG [15, 16, 18, 19, 20].

One advantage of logic grammars lies in the rapid prototyping capabilities of Prolog, and in particular the ability to easily add new syntactic constructs. For example, in GENLANG queries are of the form <*pattern*>`:`<*parse variable*> `==>` <*input*>, where new infix operators separate grammar elements: the *pattern* generally contains the top-level nonterminal in the grammar, the *parse variable* is a logic variable (denoted by an initial uppercase letter) to which a parse tree will be bound, and the *input* is as described below. In the parse tree, nonterminals are typically adorned with information about their cost (in number of mismatches), their location in the input, the actual primary sequence recognized, etc. Similarly, *gaps* of either unbounded (`...`) or bounded (e.g. `19...27`) extent are available in the language.

A more significant extension to DCGs, called *string variables* [16, 18], also benefits from the Prolog milieu. A string variable is a logic variable appearing in the body of a grammar rule, which stands for a string of arbitrary extent, and which may optionally have applied to it operators such as the tilde which denotes reverse complementarity. Such a feature in this domain makes it easy to specify even complex arrangements of direct and inverted repeats, such as are characteristic of secondary structure:

```
tandem_repeat ---> X, X.                stem_loop ---> X, ..., ~X.
```

```
pseudoknot ---> X, ..., Y, ~X, ..., ~Y.

attenuator ---> X, ..., ~X, ..., X.
```

(A somewhat different notation will be presented in the tutorial.) These descriptions are at a much higher level than the corresponding context-free grammars [16], and in fact most of them are even beyond context-free, yet despite their widely varying linguistic complexity they are expressed with comparable ease in this formalism. Even so, these are relatively abstract descriptions and for purposes of parsing require "real world" constraints on their length and degree of mismatch allowed in stems; we next describe the mechanism for controlling these.

Objects in GENLANG, e.g. nonterminals, can have attached to them an *attribute list* as shown in this example:

```
foo:[cost=S+2*C] ---> "atg", ...:[step=3,S=size], bar:[size<50,C=cost].
```

Here, the control attribute `step=3` specifies that the gap (`...`) is to increase in increments of 3; the constraint attribute `size<50` keeps the span of the nonterminal `bar` under 50; the specification attribute `cost=S+2*C` redefines the cost of the nonterminal `foo` to an arithmetic function of the size of the gap and the cost of the `bar`; and the assignment attributes `S=size` and `C=cost` serve to bind those variables. (The default cost of `foo` would have been the number of mismatches in the `"atg"` plus those within `bar`.)

Attributes are managed by way of additional "hidden parameters" in the implementation of the grammar. Just as the difference lists serve to unburden the grammar designer of the low-level programming involved in input list management, a total of ten hidden parameters now hide from the user such details as the accumulated cost of parse trees and the cost thresholds applied by the grammar, additional constraints on the size ranges of individual elements in the grammar, and the parse tree itself.

Gaps represent regions that are "skipped over", but in fact it is the gaps that do the skipping—they constitute the multiple, embedded search engines of a typical grammar, and the source of most of its non-determinism—and so careful attention to their implementation has been necessary. One important feature is *delayed (lazy) evaluation:* gaps encountered in the course of a parse are "packaged" and passed down the parse tree, and are not actually evaluated until they in turn encounter some feature with which they may combine for more efficient evaluation. For example, the combination of a "lazy gap" with a string of bases might under the right circumstances allow the string simply to be looked up in a hash table, rather than searched for in the primary sequence. GENLANG does, at the option of the user, hash its input into k-tuples of varying sizes, and permits many hundred-fold more efficient recognition of features such as direct repeats.

The implementation of lazy gaps also lends itself to finer control over the search strategy used by the parser. The logic-based parser is ordinarily breadth-first on the *input*, in the sense that all applicable rules will be tried at every position in a parse before moving on to the next position following a gap. However, a *lazy* gap will be passed to the first applicable rule, and that rule will be tried in every possible position permitted by the gap, before the gap is passed on to the next applicable rule—in other words, depth-first search on the input. The search style in GENLANG can be controlled either at a global level, or locally through the use of the attributes `deep`, `wide` or `best` (the latter performing best-first search within a defined range).

A rule such as `foo:[consensus] ---> "gat" | "gaa" | "gta" | ...` would ordinarily succeed upon recognizing any of the disjuncts on the input string (or, any of the disjuncts with mismatches allowed up to the cost threshold established higher in the parse tree). However, with the `consensus` attribute, the disjuncts are treated as exemplars in the calculation, at compile time, of a weight matrix which now contributes the cost at this point of the parse. Moreover, the order in which the parser examines positions in the input string is reordered at compile time, so that the most "informative" positions—that is, those for which inappropriate input will most rapidly cause the nonterminal to exceed its cost threshold and thus fail—are examined first, for optimal efficiency. Base frequency data may also be entered into the grammar in tabular form, based on published data; however, there is little or no performance penalty at compile time even for large lists of exemplars, and the former technique allows the user to enter new data freely, postulate classes by dividing exemplars among several nonterminals, etc. There are several methods available for calculating costs from base frequency data (e.g. the attribute `cost=neglog` uses a negative logarithm of base frequency), which are user-definable as well.

We have used GENLANG to develop grammars for a number of higher-order patterns in sequence data, including transfer RNAs, described in the tutorial. Perhaps the most active current area of higher-order pattern recognition in biology is that of finding protein-encoding genes. In actual practice, this activity seems to devolve to two problems: recognizing splice sites, and distinguishing coding regions (exons) from noncoding regions. To a large degree these problems are duals of each other, in that completely solving one would essentially provide a solution to the other. Until recently, however, they were addressed separately; recognition of splice sites was attempted using techniques such as weight matrices and neural nets [2, 10, 11, 13, 21], while a variety of statistical techniques, beginning with codon usage frequencies and extending also to Markov chain models and connectionist methods, have been applied to the identification of coding regions [1, 3, 4, 9, 22]. While the results of these studies have been increasingly impressive, using these distinct approaches in isolation may never be completely satisfactory.

The most successful such system, the multiple-sensor neural net *Grail* [22], in fact uses a *combination* of evidence from seven previously-described algorithms to identify about 90% of large exons with about one in six false positives. The trend, in fact, is toward layered or rule-based architectures which combine evidence about not only coding regions but splice sites as well, to better delineate the former and to reduce the combinatoric possibilities of the latter [5, 7, 8, 12]. These systems owe their success to a hierarchical organization of evidence based on statistical measures, and above all to their ability to consider that evidence in mutual context. We have implemented gene finder based on a simple gene grammar, as described in the tutorial, which has been surprisingly successful in comparison with highly-specialized procedural gene finders.

References

[1] M. Borodovsky. Genmark: System for predicting protein coding regions. Technical Report Version 1.1, Georgia Tech School of Applied Biology and Office of Information Technology, genmark@ford.gatech.edu, 1992.

[2] S. Brunak, J. Engelbrecht, and S. Knudsen. Prediction of human mRNA donor and acceptor sites from the DNA sequence. *J. Mol. Biol.*, 220:49–65, 1991.

[3] G. Fichant and C. Gautier. Statistical method for predicting protein coding regions in nucleic acid sequences. *CABIOS*, 3:287–295, 1987.

[4] J. W. Fickett. Recognition of protein coding regions in DNA sequences. *Nucleic Acids Res.*, 10:5303–5318, 1982.

[5] C. A. Fields and C. A. Soderlund. gm: a practical tool for automating DNA sequence analysis. *CABIOS*, 6(3):263–270, 1990.

[6] K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[7] M.S. Gelfand. Computer prediction of the exon-intron structure of mammalian pre-mRNAs. *Nucleic Acids Res.*, 18:5865–5869, 1990.

[8] R. Guigo, S. Knudsen, N. Drake, and T. Smith. Prediction of gene structure. *J. Mol. Biol.*, 226:141–157, 1992.

[9] A. K. Konopka. Towards mapping functional domains in indiscriminantly sequenced nucleic acids: A computer approach. In *Structure and Methods*, pages 113–125. Adenine Press, 1990.

[10] M. Kudo, Y. Lida, and M. Shimbo. Syntactic pattern analysis of 5' splice site sequences of mRNA precursors in higher eukaryotic genes. *CABIOS*, 3:319–324, 1987.

[11] A. S. Lapedes, C. Barnes, C. Burks, R. M. Farber, and K. M. Sirotkin. Application of neural networks and other machine learning algorithms to DNA sequence analysis. In G. I. Bell and T. G. Marr, editors, *Computers and DNA*. Addison-Wesley, Redwood City CA, 1988.

[12] R. Legouis et al. The candidate gene for the X-linked Kallmann Syndrome encodes a protein related to adhesion molecules. *Cell*, 67:423–435, 1991.

[13] K. Nakata, M. Kanehisa, and C. DeLisi. Prediction of splice junctions in mRNA. *Nucleic Acids Res.*, 13:5327–5340, 1985.

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

[15] D. B. Searls. Representing genetic information with formal grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 386–391. American Association for Artificial Intelligence, 1988.

[16] 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*, pages 189–208. MIT Press, 1989.

[17] D. B. Searls. Signal processing with logic grammars. *Intelligent Systems Review*, 1(4):67–88, 1989.

[18] D. B. Searls. String variable grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming*, 1995. In press.

[19] 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.

[20] D. B. Searls and M. O. Noordewier. Pattern-matching search of DNA sequences using logic grammars. In *Proceedings of the Conference on Artificial Intelligence Applications*, pages 3–9. IEEE, 1991.

[21] R. Staden. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res.*, 12:505–519, 1984.

[22] E. C. Uberbacher and R. J. Mural. Locating protein-coding regions in human DNA sequences by a multiple sensor-neural network approach. *Proc. Nat. Acad. Sci. USA*, 88:11261–11265, 1991.

Logic Grammars (I)

- *definite clause grammar* is an alternative Prolog notation, creating a recursive-descent parser:

```
sentence --> noun_phrase, verb_phrase.
noun_phrase --> determiner, modified_noun.
modified_noun --> noun |
   adjective, modified_noun.
determiner --> [the].      noun --> [man].
```

- we use a derivation operator to denote a parse query (which may be over an initial substring):

```
| ?- sentence ==> [the,old,man,saw,me].
yes

| ?- noun_phrase ==> [the,old,man,saw,me]/R.
R = [saw,me]
```

Logic Grammars (II)

- logic grammars also allow parameters, and embedded Prolog code (in curly braces)

```
sum(S) --> [X], sum(S0), {S is X+S0}.
sum(0) --> [].
```

- terminal strings may also be *replaced* on the input string, when they appear after the non-terminal on the left hand side of a rule:

```
inversion, [Y] --> [X], {Y is -X}, inversion.
inversion --> [].
```

- this allows grammars to alter the input, and to perform wholesale operations on strings, when invoked in the following pattern:

```
inversion ==> Input/Output.
```

Logic Grammars (III)

- a DCG translator adds “input/output” parameters called *difference lists* to a grammar rule, converting it into a Prolog rule that accepts an input string and returns the “leftover”

```
a --> b, c.            a(S0/S) :- b(S0/S1),
                                  c(S1/S).
```

- terminal elements are shown in Prolog lists (or double-quoted character strings), which are translated to a form that consumes input

```
a --> [x,y,z].         a(S0/S) :- S0=[x,y,z|S].
```

- parameters and attached code are easily added

```
a(X) --> b(X),         a(X,S0/S) :- b(X,S0/S),
         {Code}.                    Code.
```

A Gene Grammar (I)

- genes can be described hierarchically, with interspersed “gaps” (bounded or unbounded):

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

- terminal elements are double-quoted DNA bases:

```
cat_box --> pyrimidine, "caat".
tata_box --> "tata", base, "a".
cap_site --> "ac".
base --> purine | pyrimidine.
purine --> "g" | "a".
pyrimidine --> "t" | "c".
```

A Gene Grammar (III)

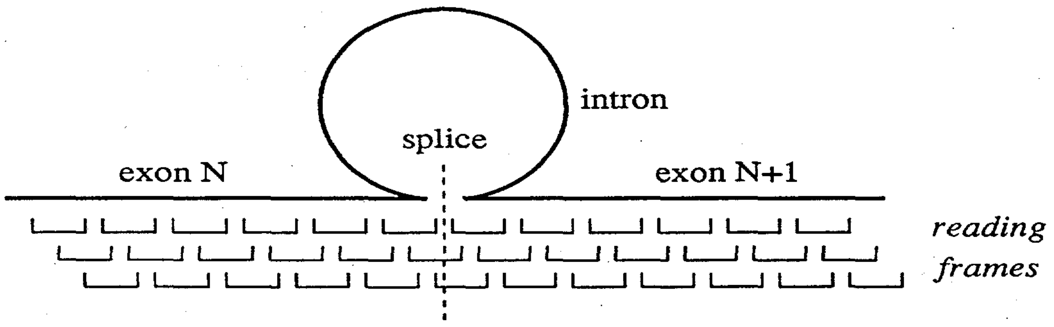
- to write a grammar for a protein coding region, we first specify codons (including stop codons)

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

- exons and translated regions are defined recursively:

```
xlate([met|RestAAs]) --> codon(met),
  rest_xlate(RestAAs), stop_codon.
rest_xlate(AAs) --> exon(AAs).
rest_xlate(AAs) --> exon(AAs1), intron,
  rest_xlate(AAs2), {append(AAs1,AAs2,AAs)}.
exon([]) --> [].
exon([H|T]) --> codon(H), exon(T).
```

A Gene Grammar (IV)



- introns are handled using a context-sensitive rule to account for splices that straddle any reading frame:

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

splice --> "gt", ..., "ag".
```

String Variable Grammar (I)

- in a logic grammar framework, we can extend context-free grammars to allow variables on the right-hand sides of rules, specifying substrings of tokens to which arbitrary *substitutions* are applied

```
[]+_ --> [].
[H|T]+F --> F:H, T+F.

[]-_ --> [].
[H|T]-F --> T-F, F:H.

1:X --> [X].     % identity substitution

palindrome --> X+1, X-1.

copy --> X+1, X+1.
```

- thus palindrome and copy languages appear similar

String Variable Grammar (II)

- SVGs handle arbitrary numbers of copies (beyond the capability of TAGs), and many other non-context-free formal languages, e.g. $\{a^n b^n c^n \mid n \geq 0\}$:

```
F:_ --> [F].      % functor substitution

aNbNcN --> N+a, N+b, N+c.
```

- complementarity substitutions come in "flavors":

```
d:t --> [a].    d:g --> [c].    % DNA/DNA
d:a --> [t].    d:c --> [g].    % subst'n

r:u --> [a].    r:g --> [c].    % RNA/DNA
r:a --> [t].    r:c --> [g].    % subst'n

s:u --> [a].    s:g --> [c].    % secondary
s:a --> [u].    s:c --> [g].    % structure
s:g --> [u].    s:u --> [g].    % (non-WC)
```

String Variable Grammar (III)

- some biological SVGs:

```
stem_and_loop --> Stem+1, Loop+1, Stem-s.

pseudoknot --> X+1, Y+1, X-s, Y-s.

attenuator --> X+1, X-s, X+1.

tRNA --> W+1, X+1, X-s,
    Y+1, Y-s, Z+1, Z-s, W-s.
```

- recursion and string variables as parameters:

```
cloverleaf --> X+1, leaves, X-s.
leaves --> Y+1, Y-s, leaves | [].

cluster(X) --> [] | (X+1 | X-d), cluster(X).
```

String Variable Grammar (IV)

- used this way, SVGs lie strictly between context-free and indexed grammers in generative power
- allowing composition of string variable substitutions increases their expressive power, e.g. $\{a^{2^n} \mid n \geq 0\}$:

```
2:X --> [X,X].    % doubling substitution

double(X) --> X+1 | double(X+2).
```

- this allows multi-level grammars:

```
p:'A' --> [g,c], ([a] | [g] | [c] | [u]).
p:'B' --> [g,a], ([c] | [u]).
   % etc...       % protein substitution

protein(X) --> rna(X+p).
rna(X) --> dna(X-r).
dna(X) --> X-d.
```

String Variable Grammar (V)

- left-hand-side string variables are useful for modelling mutation and evolution

```
duplication, X+1, X+1 --> X+1.
inversion, X-d --> X+1.
transposition, Y+1, X+1 --> X+1, Y+1.

evolution --> [] | event, evolution.
event, X+1 --> X+1,
  (duplication | inversion | transposition).
```

- in fact, additional biological knowledge can be incorporated, e.g. the fact that repeats S may give rise to excision of the intervening sequence X as a circle C (created here using recursive derivation)

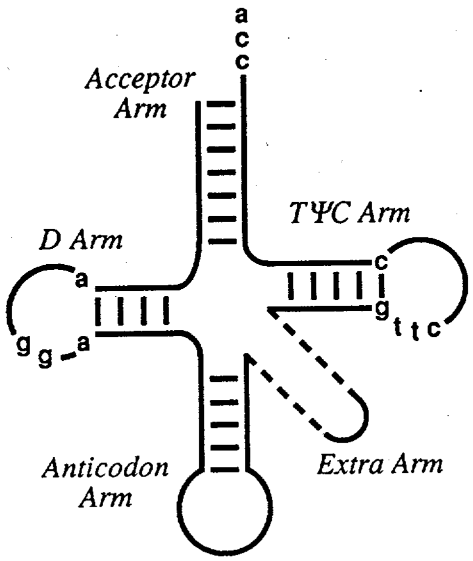
```
excision(C), S+1 --> S+1, X+1, S+1,
   {(S,X)==>C/C}.
```

String Variable Grammar (VI)

- gene expression can also be modelled by means of such "side-effecting" grammars (using an additional predicate end that ensures the entire string is consumed), invoked as `expression ==> DNA/Protein`

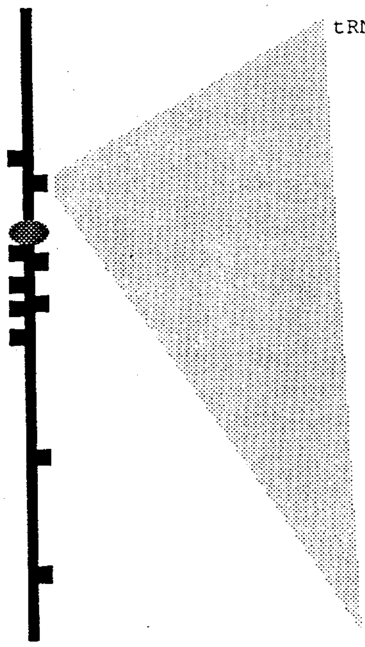
```
expression -->
   transcription, processing, translation.

transcription, X-r --> X+1, end.

processing, X+1 --> X+1, [g,u], _+1, [a,g],
   processing.
processing --> [].

translation, ['M'|X]+1 --> _+1, [a,u,g],
   X+p, termination, _+1, end.
termination --> [u,g,a] | [u,a,g] | [u,a,a].
```

Transfer RNA Syntax



```
tRNA → Stem#7,
       "t", base,
       d_arm, base,
       anticodon_arm,
       extra_arm,
       t_psi_c_arm,
      ~Stem$1,
       acceptor_arm.
d_arm → Stem#4,
       "a", purine, 1...3,
       "gg", 1...3,
       "a", 0...1,
      ~Stem$1.
          .
          .
          .
```

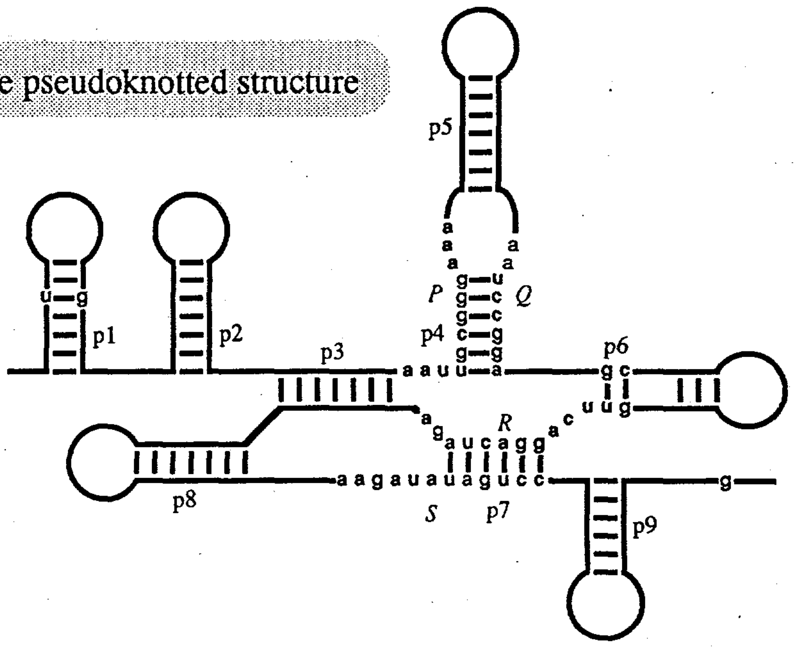
Yeast ChIII tRNA Parsing



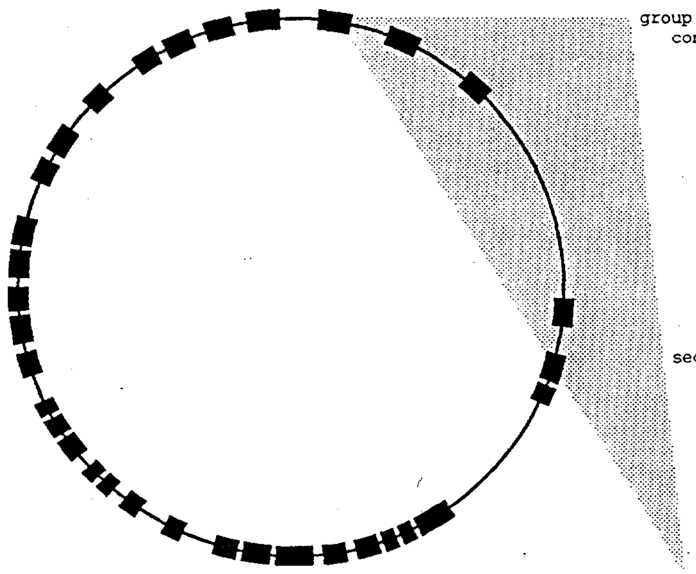
```
tRNA("ttg"): span=90470/90583
   "ggttgtt": span=90470/90477 cost=0
   turn: list="tg"
   D arm: span=90479/90496
      stem: list="gcc"
      loop: list="gagcggtctaa" cost=0
     ~stem: list="ggc" cost=0
   base: list="g"
   anticodon arm: span=90497/90546
      "cctga": span=90497/90502
      loop: cost=0
         pre_codon: list="tt"
         ~codon: list="caa"
         purine: list="g"
         intron: span=90508/90540
            ...: size=7
            "ttg": span=90515/90518 cost=0
            ...: size=22
         base: list="c"
     ~"cctga": span=90541/90546 cost=0
   extra arm: span=90546/90559
      ...: size=13
   TpsiC arm: span=90559/90576
      stem: list="aagag"
      loop: list="ttcgaat" cost=6
     ~stem: list="ctctt" cost=0
  ~"ggttgtt": span=90576/90583 cost=0.5
```

Group I Introns

• note pseudoknotted structure

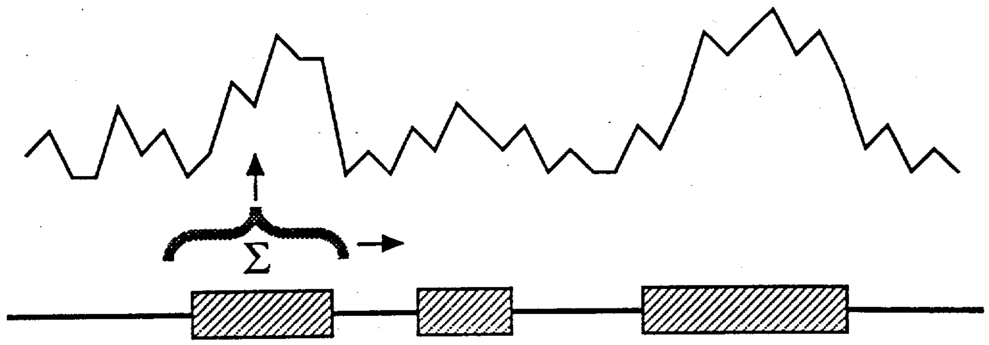


Fungal Mitochondrion Intron Parsing



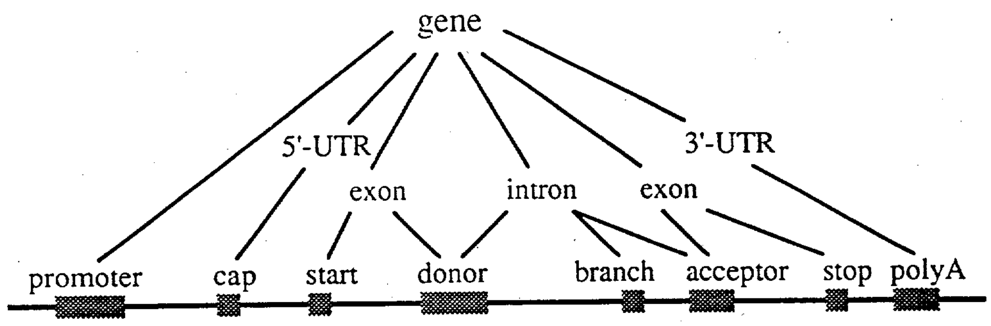
```
group I intron: span=3237/4660
   conserved regions: span=3237/4638
      P region: span=3237/3249 cost=297
                list="aatttcaaaaac"
      P-Q loop: span=3249/3283
        ...: size=34
      Q region: span=3283/3293 cost=230
                list="gatttgaagc"
      ...: size=135
      R region: span=3428/3442 cost=258
                list="gttcaacgactaag"
      R-S loop: span=3442/4626
        ...: size=82
         ORF: span=3524/4508 size=984
        ...: size=118
      S region: span=4626/4638 cost=134
                list="aagacatagtct"
   secondary struture: span=3230/4660
      p3: span=3230/3235 list="ttggg"
      p4: span=3240/3246 list="ttcaaa"
      p5/~p5: span=3249/3283
      ~p4: span=3285/3291 list="tttgaa"
      p6/~p6: span=3290/3431
      p7: span=3435/3440 list="gacta"
      ~p3: span=3476/3481 list="cccaa"
      p8/~p8: span=3481/4625
      ~p7: span=4632/4637 list="tagtc"
      p9/~p9: span=4640/4660
```

Compositional Gene Search



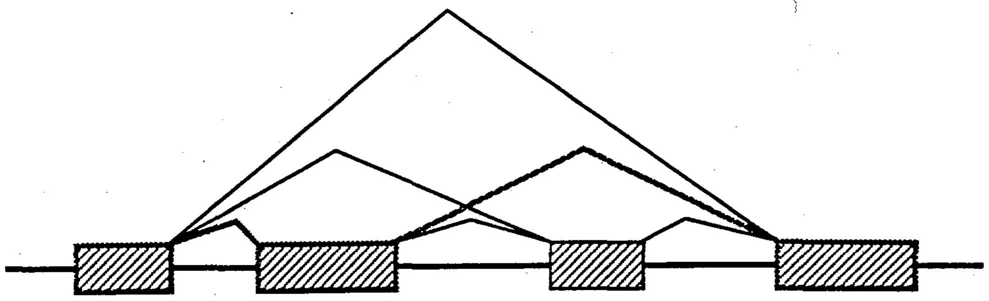
- "Gene Search by Content" (Staden)
  - open reading frames
  - codon preference
- Fickett's Algorithm (Position Asymmetry)
- Hextuple Frequencies
- Information-Theoretic Approaches
- GRAIL Coding Recognition Module

Syntactic Gene Search



- "Gene Search by Signal" (Staden)
- Hybrid Systems
  - GeneID
  - GeneParser
  - GRAIL follow-ons
- GenLang Linguistic Parser

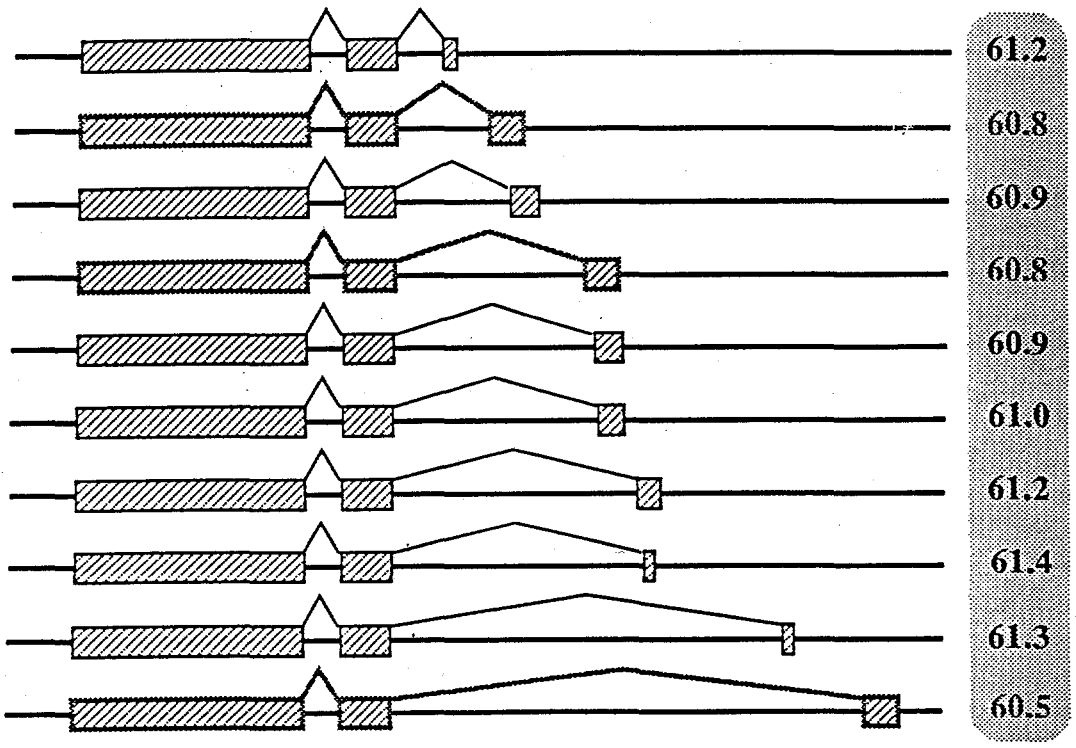
Gene Search Combinatorics



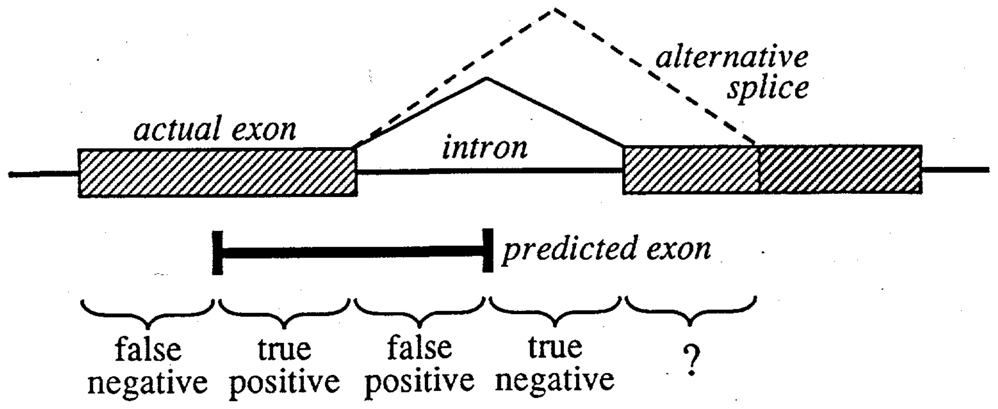
$$\textit{Potential Genes} = 2^{\textit{Potential Internal Exons}}$$

- GeneParser: dynamic programming
- GeneID: clustering into exon classes
- GRAIL: bottom-up heuristic evaluation
- GenLang: top-down chart parsing and successive approximation

Nondeterministic Parsing



Gene Prediction Metrics

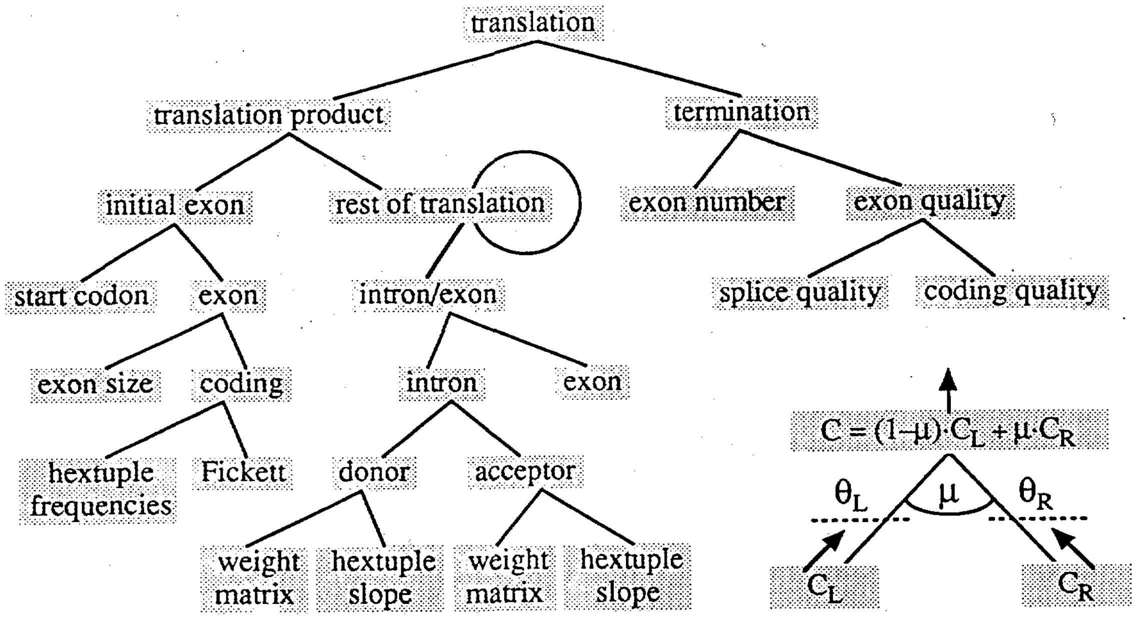


Sensitivity = TP/(TP+FN) **Specificity** = TN/(TN+FP)

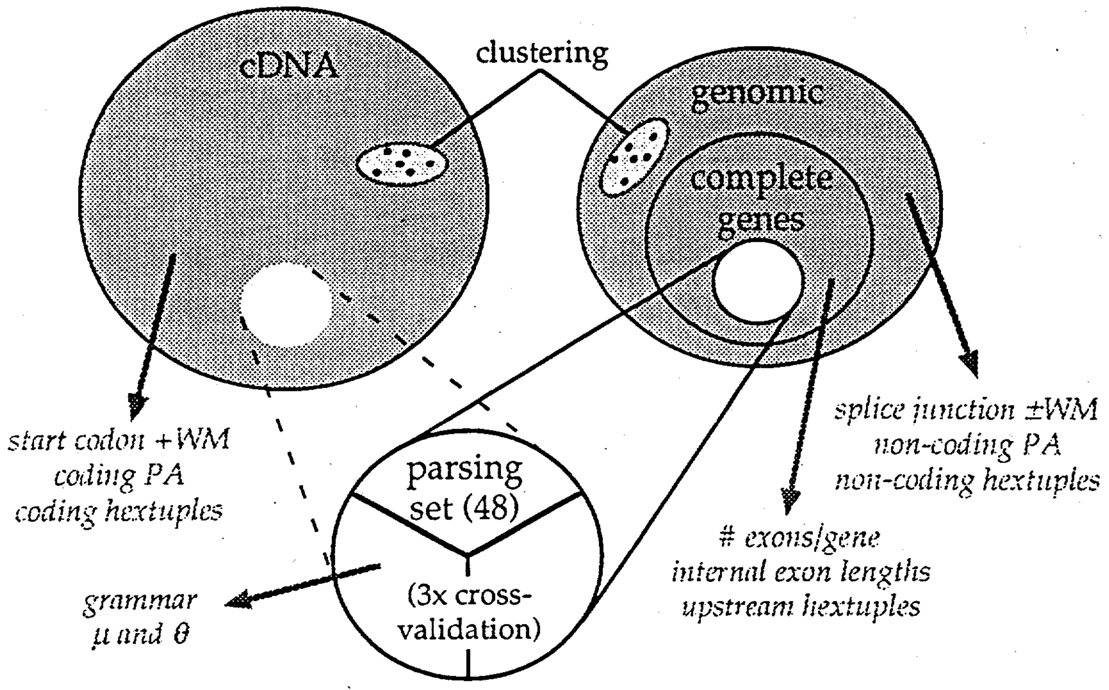
Positive Predictive Value = TP/(TP+FP)

$$\textbf{2x2 Correlation Coefficient} = \frac{(TP \cdot TN - FP \cdot FN)}{\sqrt{(TP+FP)(FP+TN)(TN+FN)(FN+TP)}}$$

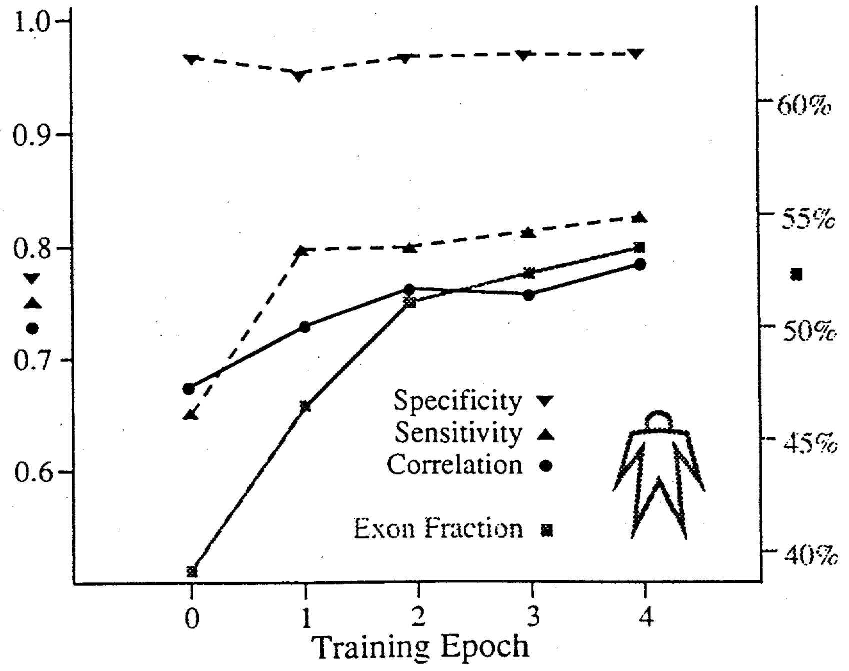
Parse Tree and Cost Mechanism



Training/Test Sets



Syntactic Training



Relative Performance

	🧍	🪱	🐝	🌼
Correlation	.76±.17	.76±.19	.83±.19	.85±.12
Sensitivity	.83	.81	.86	.86
Specificity	.95	.96	.97	.98
PPV	.80	.82	.94	.92
Exons	50%	56%	56%	58%
Genes	13%	13%	29%	23%
Epoch	1st	4th	3rd	3rd
Rank	24th	27th	10th	13th

- Improved results on non-vertebrate sets
- Also more correct parses, earlier in rank ordering

Inter-Species Comparison

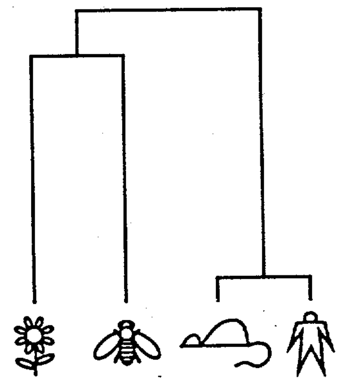
Correlation Coefficient / Exon Fraction

train \ test	🧍	🪱	🐝	🌼
🧍	.75/51%	.72/52%	.31/17%	.27/16%
🪱	.75/48%	.77/55%	.63/28%	.55/26%
🐝	.33/10%	.36/14%	.83/55%	.47/26%
🌼	.21/5%	.23/6%	.61/35%	.87/67%

- Compares languages (*sets* of strings), not strings

$$Distance_{12} \equiv CC_{11} \cdot CC_{22} - CC_{12} \cdot CC_{21}$$

UPGMA Dendrogram



Early Work

- the earliest references to linguistics of DNA were simply the free use of the linguistic metaphor by biologists, e.g. in naming processes such as transcription, translation, proofreading, editing, etc., and the occasional purely philosophical treatise by eminences such as Chargaff

- the first investigations by linguists (e.g. Shanon) were skeptical, based on a very limited view of the genetic code and protein translation as devoid of context

- Jimenez-Montano and co-workers made first use of grammars, but essentially only as data structures for analysis of information-theoretic complexity of protein sequences

- Brendel and Busse proposed a simple automaton for gene expression, in what was the first demonstration of any faith in the utility of linguistic formalisms, but they never pursued the work further

[1] E. Chargaff. Preface to a grammar of biology. *Science*, 172:637–642, 1971.

[2] B. Shanon. The genetic code and human language. *Synthese*, 39:401–415, 1978.

[3] W. Ebeling and M. A. Jimenez-Montano. On grammars, complexity, and information measures of biological macromolecules. *Math. Biosci.*, 52:53–71, 1980.

[4] M. A. Jimenez-Montano. On the syntactic structure of protein sequences and the concept of grammar complexity. *Bull. Math. Biol.*, 46(4):641–659, 1980.

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

Vocabularies and Codes

- Trifinov and colleagues (Weizmann Institute) coined the use of the term "linguistics" in relation to DNA, but use it to refer to vocabularies, or tendencies of certain words to appear with different frequencies in certain contexts, and more recently to the notion of overlapping "codes" or signals of various types which overload the DNA and increase its information-theoretic complexity

[1] E.N. Trifinov and V. Brendel. *Gnomic — A Dictionary of Genetic Codes.* Balaban Publishers, Rehovot–Philadelphia, 1986.

[2] 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.

[3] J.S. Beckmann, V. Brendel, and E.N. Trifinov. Intervening sequences exhibit distinct vocabulary. *J. Biomol. Struct. Dynamics*, 4:391–400, 1986.

[4] E.N. Trifinov. Nucleotide sequences as language: morphological classes of words. In H.H. Bock, editor, *Classification and Related Methods of Data Analysis*, pages 57–64. Elsevier (North Holland), 1988.

[5] E.N. Trifinov. The multiple codes of nucleotide sequences. *Bull. Math. Biol.*, 51:417–432, 1989.

[6] S. Pietrokovski, J. Hirshon, and E.N. Trifinov. Linguistic measure of taxonomic and functional relatedness of nucleotide sequences. *J. Biomol. Struct. Dynamics*, 7:1251–1268, 1990.

[7] S. Pietrokovski and E.N. Trifinov. Imported sequences in the mitochondrial yeast genome identified by nucleotide linguistics. *Gene*, 122:129–137, 1992.

[8] E.N. Trifinov. DNA as a language. In H.A. Lim, J. Fickett, C.R. Cantor, and R.J. Robbins, editors, *Proceedings of the 2nd International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 103–110. World Scientific, 1993.

Vocabularies and Statistics

- the vocabulary-based approach has been picked up by mathematicians (notably Gelfand and Pevzner) studying base compositions using Markov models, etc.

[1] 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 frequency of occurrence of words. *J. Biomol. Struct. Dynamics*, 6:1013–1026, 1989.

[2] 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. Dynamics*, 6:1027–1038, 1989.

[3] M.S. Gelfand, C.G. Kozhukhin, and P. A. Pevzner. Extendable words in nucleotide sequences. *CABIOS*, 8(2):129–135, 1992.

[4] M. S. Gelfand. Genetic language: metaphore or analogy? *BioSystems*, 30(1–3):277–288, 1993.

Stochastic Grammars

- David Haussler and colleagues (U.C. Santa Cruz) study RNA structure using stochastic context-free grammars as a generalization of hidden Markov models

[1] Y. Sakakibara, M. Brown, I.S. Mian, R. Underwood, and D. Haussler. Stochastic context-free grammars for modeling RNA. In *Proceedings of the Hawaii International Conference on System Sciences*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[2] Y. Sakakibara, M. Brown, R. Hughey, I.S. Mian, K. Sjölander, R.C. Underwood, and D. Haussler. Recent methods for RNA modeling using stochastic context-free grammars. In *Proceedings of the Asilomar Conference on Combinatorial Pattern Matching*, New York, NY, 1994. Springer-Verlag. In press.

Regulatory Grammars

- the work of Julio Collado (now at U. of Mexico, Cuernavaca) is based on the application of ideas from transformational grammar to account for variation in the arrays of regulatory elements associated with bacterial genes
- Collado claims a formal proof that context-free grammars are inadequate, based on the pumping lemma and the observation that order of regulatory factors are independent of the genes they control [2]
- to date, no practical applications have been developed

[1] J. Collado-Vides. A transformational-grammar approach to the study of the regulation of gene expression. *J. Theor. Biol.*, 136:403–425, 1989.

[2] J. Collado-Vides. The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars. *CABIOS*, 7(3):321–326, 1991.

[3] J. Collado-Vides. A syntactic representation of units of genetic information — a syntax of units of genetic information. *J. Theor. Biol.*, 148:401–429, 1991.

[4] J. Collado-Vides. Grammatical model of the regulation of gene expression. *Proc. Nat. Acad. Sci. USA*, 89(20):9405–9409, 1992.

[5] J. Collado-Vides. The elements for a classification of units of genetic information with a combinatorial component. *J. Theor. Biol.*, 163(4):527–548, 1993.

[6] J. Collado-Vides. A linguistic representation of the regulation of transcription initiation. I. An ordered array of complex symbols with distinctive features. *Biosystems*, 29(2–3):87–104, 1993.

[7] J. Collado-Vides. A linguistic representation of the regulation of transcription initiation. II. Distinctive features of sigma 70 promoters and their regulatory binding sites. *Biosystems*, 29(2–3):105–128, 1993.

Splicing Systems

- Splicing Systems are formal structures developed by Tom Head (now at SUNY Binghampton) and others to study the generative capacity of restricting and religating sets of double-stranded linear and/or circular DNA molecules

[1] K. Culik II and T. Harju. The regularity of splicing systems and DNA. In *16 International Colloquium on Automata Languages and Programming (Lecture Notes in Computer Science 372)*, pages 222–233. Springer-Verlag, Berlin, 1989.

[2] K. Culik II and T. Harju. Splicing semigroups of dominoes and DNA. *Discrete Applied Mathematics*, 31:261–277, 1991.

[3] K.L. Denninghoff and R.W. Gatterdam. On the undecidability of splicing systems. *International Journal of Computer Mathematics*, 27:133–145, 1989.

[4] R.W. Gatterdam. Splicing systems and regularity. *International Journal of Computer Mathematics*, 31:63–67, 1989.

[5] R.W. Gatterdam. Algorithms for splicing systemsy. *SIAM Journal of Computing*, 21:507–520, 1992.

[6] 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.

[7] T. Head. Splicing schemes and DNA. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer Systems — Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, pages 335–342. Springer-Verlag, Berlin, 1992. (also in *Nanobiology* 1:335-342, 1992).

[8] R. Siromoney, K.G. Subramanian, and V.R. Dare. Circular DNA and splicing systems. In *Parallel Image Analysis (Lecture Notes in Computer Science 654)*, pages 260–273. Springer-Verlag, Berlin, 1992.

[9] R. Siromoney, K.G. Subramanian, and V.R. Dare. On identifying DNA splicing systems from examples. In *(Lecture Notes in Artificial Intelligence 642)*, pages 305–319. Springer-Verlag, Berlin, 1992.

Structure and Search

- my own work is based on:
  - the study of structural phenomena in biological sequences and their classification in terms of formal language theory
  - the development of domain-specific grammar/parser systems and their use in syntactic pattern recognition tasks such as gene search

[1] D. B. Searls. Representing genetic information with formal grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 386–391. American Association for Artificial Intelligence, 1988.

[2] 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*, pages 189–208. MIT Press, 1989.

[3] D. B. Searls and M. O. Noordewier. Pattern-matching search of DNA sequences using logic grammars. In *Proceedings of the Conference on Artificial Intelligence Applications*, pages 3–9. IEEE, 1991.

[4] D. B. Searls. The computational linguistics of biological sequences. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, chapter 2, pages 47–120. AAAI Press, 1993.

[5] D. B. Searls and S. Dong. A syntactic pattern recognition system for DNA sequences. In H. A. Lim, J. Fickett, C. R. Cantor, and R. J. Robbins, editors, *Proceedings of the 2nd International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 89–101. World Scientific, 1993.

[6] D. B. Searls. The linguistics of DNA. *American Scientist*, 80(6):579–591, 1992.

[7] D. B. Searls. String variable grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming*, (in press).

[8] S. Dong and D. B. Searls. Gene structure prediction by linguistic methods. *Genomics*, 23:540-551, 1994