DOE/ER/61371--1

DE93 009773

EXHIBIT A

DOE Report No.	DOE/ER/61371-1
Title and Subtitle	"Foundations for a Syntatic Pattern Recognition System for Genomic DNA Sequences"
Report Period (if applicable)	1 December 1991 through 31 March 1993
Personal Author(s)	Dr. David B. Searles
Contractor's Name and Address	Trustees of the University of Pennsylvania 133 South 36th Street, Suite 300 Philadelphia, PA 19104-3246
Report Date	March 1993
DOE Sponsorship and DOE Instrument Number	**Prepared for** Department of Energy (THE U. S.) Agreement No. DE-FG02-92ER61371

Justification

Personnel

Prof. David Searls (30%). Dr. Searls will serve as principal investigator, making the major design decisions and coordinating the work of all project staff and interactions with collaborators.

Programmer/Analyst (100%). One full-time Master's level person will be the chief programmer on the project, responsible for all graphical interface development, for integration of new parser code and configuration control of the software package as a whole and associated libraries, and for expert-level grammar development in association with collaborators.

Programmer/Analyst (40%). One additional Master's level person specializing in systems programming and algorithms will devote 10% time performing systems administration tasks specific to this project, and 30% time working on the incorporation of advanced parser technology.

Programmer/Analyst (40%) One additional Master's level person specializing in databases and Prolog programming will be primarily responsible for adaptation of the system to new input formats, including interconnection, database interfaces, header processing, and ASN.1 interface, and assisting in grammar design.

Budget Assistant (15%) The departmental office will provide program management support.

Equipment and Supplies

Workstation: The SPARCstation 10 requested under this application will be used by one of the project staff (three existing SPARCstation 2's will be provided for the remainder). This significantly more powerful workstation will also be configured as a network "Parse Server" for use in off-line batch parsing of computationally intensive queries on an availability basis, and by collaborators for X-hosted remote access at any time. An annual service contract is also budgeted, in line with our current arrangements.

Remote Access Software: Software for X-based access to the parse server from Macintosh, PC, and X-terminal platforms will be necessary to promote the use of the system by collaborating biologists without easy access to workstations.

Color Laser Printer: The extensive use of scientific visualization in the proposed work will require color output, and the relatively expensive supplies associated with it, for permanent records of parses, reports, publications, and presentation transparencies, both for the staff and for collaborators.

Miscellaneous Supplies: These will include copier supplies, general office supplies, costs of mailing, etc.

Travel

Travel costs represent an annual estimate based on 2x5 person-days to Washington DC for scientific and collaborator meetings, 1x4 person-days to San Francisco for scientific conferences, and 1x2 person-days to Boston for collaborator meetings.

Project Description

1 Objectives

The goal of the proposed work is to extend, refine, and apply the Principal Investigator's research into linguistic analysis of biological sequences. This will result in the creation of a software system that will perform sophisticated pattern recognition and related functions (1) at a level of abstraction and with expressive power beyond current *general-purpose* pattern-matching systems for biological sequences; and (2) with a more uniform language, environment, and graphical user interface, and with greater flexibility, extensibility, embeddability, and ability to incorporate other algorithms, than current *special-purpose* analytic software. The specific aims to be accomplished are:

1. *Extended development of the graphical user interface and visualization tools.* A current dynamic parse visualization tool will be enhanced, and supplemented with static data visualization routines for high-level iconic depiction of parse results. A graphical interface will be implemented to support interactive grammar development and refinement in a rapid-prototyping mode.

2. *Development of embeddability "hooks" for incorporation of, and by, other algorithms.* The system will be made into a platform for the application of other algorithms in a hierarchical fashion, focusing them on regions of interest, providing a uniform environment for input, output, and parameter management, and assembling results into the grammar's structural model. The grammar system itself will also be made embeddable in other platforms where appropriate.

3. *Incorporation of advanced parser technology and application to eukaryotic gene parsing.* Current developments in island parsing, probabilistic parsing, etc. will be embedded in the system, driven by the specific practical problem of efficient recognition of protein-coding eukaryotic genes. Current statistical and heuristic gene-finding algorithms will be adapted to grammatical expression, to allow for greater flexibility and contextually structured application.

4. *Extension of the repertoire of input formats accepted and header information processed by the parser.* The current GenBank flat file entry parser will be extended to handle a variety of other formats, and to extract additional information from features tables for graphical depiction and high-level parsing. Facilities for transparent connection to relational databases and ASN.1-formatted data streams will also be developed.

5. *Extension of the grammar system to encompass protein sequence at multiple levels.* The parser will be extended to accept single-letter protein code as its primary sequence, with which motifs will be described. Longer-term goals include the development of secondary structure grammars and potentially even the description of tertiary structures using coordinate grammars.

6. *Collaborations aimed at specific biological and computational problems.* In order to further drive the development of the system in biologically relevant directions, collaborations with biologists for grammar development, and with computational biologists for parser development, will be undertaken. A facility for remote access to the parser will be provided.

7. *Distribution and promotion of the software and associated libraries.* Periodic releases of the software to any interested parties will be accompanied by full documentation and a reasonable

level of support, in particular in the development of new grammars. Grammars will be maintained in a central, publicly-accessible repository of biological feature specifications, for use either with the parser or with other programs.

2 Background

The task of describing and searching for substrings of interest in long biological sequences has assumed a central role in experimental molecular biology. Here, we very briefly review current approaches, and then present technical background for the methodology proposed in this application.

2.1 Search in Molecular Biology

The task of *search* in biological sequences and sequence databases can be viewed in terms of variations on the basic problem of *string search*, or the discovery of a specified substring in a much longer target string. Well established variants of this problem, all of which are useful in molecular biology, include: *(1) information retrieval* search, which typically enhances the basic techniques with the ability to specify boolean combinations of substrings that work against large, flat databases of text to retrieve relevant blocks of information; *(2) similarity* search, in which dynamic programming and other sophisticated computational techniques are used to retrieve examples of sequence data that may share an evolutionary relationship to part or all of a given string, often by virtue of some mutational model embodied in the algorithm; and *(3) pattern-matching* search, which involves a more or less abstracted description or model of a substring or class of substrings of interest, as opposed to an actual instance of such a string for comparison. We consider the latter two in particular.

2.1.1 Regular Expression and Similarity Search

By and large, similarity and pattern-matching search can be distinguished by their concept of the object of the search: in similarity search, a concrete *example* is given to a special-purpose algorithm that performs a single uniform style of search, whereas in pattern-matching search an abstract *description* of a *set* of strings is given to a general-purpose algorithm whose exact computational behavior depends on the nature of the specification. (Thus, the latter specification has more of the flavor of a simple computer program than does the example string supplied to the former.)

Examples of regular expression pattern-matching search and similarity search software abound, typified in the IntelliGenetics package by the *QUEST* and *IFIND* programs [26], respectively, and in GCG by *FindPattern* and *FastA* [13]. Since full alignment dynamic programming algorithms are inherently $O(n^2)$, these similarity search programs use speedups based, for example, on hash-table lookup of k-tuples instead of individual bases; more recent enhancements to similarity search are aimed at even greater efficiency, as in *BLAST* [3], which sacrifices some sensitivity but in a statistically well-founded way. On the other hand, enhancements to the inherently $O(n)$ regular expression search algorithm generally involve attempts to increase the expressive power of regular expressions, particularly with features important to the domain such as inverted and direct repeats [1, 62], without sacrificing efficiency. As we will see, adding such features is not simply a convenience, for "native" regular expressions are unable to express unbounded repeats, and repeats of any length are extremely awkward. Generally end-users of these regular expression matchers do not attempt very complex expressions, since their inherently "flat" format makes it difficult to build up layered specifications with many interdependencies. Nevertheless there has been much recent interest in just such deep patterns, for example in certain forms of protein *motifs*, which make use of enhanced regular expression pattern-matching [33].

2.1.2 Inductive Pattern Matching

Pattern specifications based on *weight matrices* [64, 68] or *connectionist* techniques [37, 61, 65] have been used extensively in molecular biology as well. Here the distinction between similarity and pattern-matching search blurs. These methods can be viewed as similarity search in which a number of exemplars, rather than a single search string, are supplied. On the other hand, they can be viewed in terms of a pattern built up inductively. In the case of weight matrices, that pattern may be abstracted as a consensus sequence, and the matrix can be considered to be simply a subtler expression of the latter. There may be a much deeper model in the case of neural network recognizers, but the structure of that model is typically far from apparent in the derived "specification". These techniques have the great advantage of being able to "learn" from properly presented data, but the relative difficulty of inserting and/or extracting any hypothesized *structure* in their patterns limits their utility in many contexts.

2.1.3 Special Purpose Pattern-Matchers

A number of programs have been written that detect what might be termed higher-order patterns, but by and large these are specific to the purpose and may not be easily generalizable. Typical of these are programs to find transfer RNAs in genomic sequences, of which the most successful has been one developed at Los Alamos [16]. This was able to fairly reliably find known tRNA sequences and when run over all of GenBank found a large number of previously unknown potential tRNAs, many of which are thought to be authentic. This is a procedural program which follows a flow chart of activities ranging from applying weight matrices for known conserved sequences to analyzing the potential for secondary structure. The latter source of information about the tRNA molecule is interesting in that it represents *dependencies* between positions in the primary sequence rather than outright conserved sequence. This aspect of tRNA was emphasized in another pattern-matcher [22], which could also be generalized to other recurring secondary structures such as autocatalytic introns; the pattern descriptors in this case were regular expression-like.

2.1.4 Gene Finding

Perhaps the most active current area of higher-order pattern recognition in biology is that of "gene finding". In actual practice, this activity seems to devolve to two problems: recognizing signal sequences, in particular 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 [9, 30, 32, 42, 62], 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 [5, 15, 17, 29, 65]. 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* [65], 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 [19, 23, 25, 35]. Systems such as *gm* [19] and *GeneId* [25], which along with Grail are the first to find practical use, owe their relative success to a hierarchical organization of evidence based on statistical measures, and above all to their ability to consider that evidence in mutual context. Current work on the Grail system,

in fact, involves the incorporation of "syntactic" rules in a hierarchical blackboard system.

2.2 Linguistic Approaches

Abstract, declarative, hierarchical descriptions of sets of strings are precisely the *raison d'être* of the science of computational linguistics, and of the study of formal grammars. Recently a number of studies have adopted a "linguistic" view of DNA sequences. Most of this work has involved examinations of the occurrences of "words" in DNA in what is essentially an information-theoretic approach [14, 27], or using statistical analyses of vocabularies in the tradition of comparative linguistics [6, 47, 48]. These approaches, however, can be distinguished from a more recent approach to linguistics, pioneered by Noam Chomsky, which attempts to study higher-order phenomena in languages. Such an approach has been taken by only a few authors [7, 12, 52], and apparently only the PI has pursued extensively the use of grammars as a means to accomplish generalized pattern recognition in this domain, as a form of *parsing*; this work is reviewed in the accompanying Progress Report and appendices. As background, we briefly introduce here the formal foundations and the relevant implementation techniques.

2.2.1 Formal Language 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 $\Sigma_{\mathrm{DNA}} = \{\mathrm{g}, \mathrm{c}, \mathrm{a}, \mathrm{t}\}$. 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 formally specify programming languages. 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 [51].

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.

2.2.2 Syntactic Pattern Recognition

This 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) [21]. 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 [21], which incorporate probabilities into grammars in a natural way (see ¶3.3.3). 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 PI [54, 57] and many others [21], and our initial results with SPR and the linguistic analysis of DNA (see Progress Report) suggest that they are appropriate approaches to the complexities of this domain as well.

2.2.3 Logic Grammars

Prolog is a programming language that 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 [46]. 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), so that this rule for x actually represents the two rules given previously.

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 power far beyond CFGs, due to their ability to attach parameters to nonterminals, embed arbitrary code in rule bodies, etc. It is our belief that they offer a firm foundation for a language that can be a powerful descriptive tool for molecular genetic features, and also offers a flexible analytical and control mechanism through Prolog's built-in DCG parser.

2.3 Biological Sequence Grammars

The PI has pursued this idea to the point of developing and deploying an advanced prototype system called GENLANG, implemented as an augmented DCG with additional 'C' code for input management and other speedups. This effort is described in (1) the accompanying Annual Progress Report (DOE Report Number DOE/ER/60998-3) and in more detail in (2) the users manual attached to that report as DOE Report Number DOE/ER/60998-4 and (3) in a preprint of a review article [56], previously submitted as DOE Report Number DOE/ER/60998-2 and attached to this application as an appendix. The reader is referred to these and in particular to the following points:

- a series of formal results concerning the position of the language of DNA in the Chomsky hierarchy (greater than context free) and its nondeterminism, inherent ambiguity, closure properties, etc.
- the use of these results in the design of a domain-specific grammar formalism, called *string variable grammar*, and its incorporation into GENLANG
- the much-improved syntax of the current version of GENLANG, using a uniform convention of *attribute lists* for parse control, constraints, parameters, etc.
- the adoption of delayed evaluation of gaps, permitting alternative search strategies and efficiency enhancements including the use of hash tables
- more sophisticated, hierarchical cost management techniques for imperfect matching over complex features
- a wider range of input formats, including the use of GenBank flat files (with attributes for selection of entries)
- the new graphical parse visualization tool, which allows for real-time depiction of growing parse trees and finer backtracking control
- recent results in large-scale parsing of tRNA genes, group I introns, and other higher-order structures
- the PostScript-based sequence visualization tool, RSVP, which we propose to incorporate into GENLANG

3 Research Plan

The major paragraphs below correspond to the Objectives given previously.

3.1 User Interface and Visualization

Well-designed graphical user interfaces are critical factors in user acceptance of software, and this is especially important for a system of the sophistication of GENLANG. The ultimate end users of the system are unlikely to be familiar with parsing algorithms, logic programming, and recursive grammar design, and an intuitive, task-oriented graphical interface is the first and best place to address the need to guide them through the process.

3.1.1 Parser Visualization and Control

Even for sophisticated grammar designers and logic programmers, visualization of a parse developing in real time is a great boon. It helps in debugging grammars, which otherwise must be traced through deeply recursive calls on a conventional Prolog debugger; when the grammar is portrayed in action against the backdrop of the input string itself, this is much easier. It also gives a dynamic depiction of "non-determinism traps", or points at which an inefficiently-written grammar begins thrashing on the input. For novices to grammar design, nothing is so educational as a real-time view of the procedural incarnation of their declarative grammars.

The parser visualization tool developed for GENLANG to date can be extended in many ways to make it still more practically useful. A wider range of variations in the form and size of the output will be incorporated to fit the correspondingly wide range of scales over which the parser is useful, from simple features to complex genes, gene clusters, and other assemblages. Since a common mode of use is to parse for features and then compare with the GenBank features table, we will incorporate features table annotations into the display, as opposed to the separate "info" window now presented, to make such comparisons easier. (This will be accomplished through the GenBank entry parser and its follow-ons, to be described in ¶3.4.) Currently user-defined arguments to nonterminals are shown on the graphical display, but standard attributes such as cost and size must be gleaned from the printed parse tree output, so as to avoid clutter on the display; we will extend the system so that such collateral information can be seen in a pop-up window by clicking on the nonterminal.

A facility for printed graphical output will also be developed. The current graphical form of the parse trees, now available only as rather poorly-resolved screen dumps, will be translated to PostScript for a high-resolution, appropriately packaged output format. In addition, more stylized PostScript output formats will be developed to reproduce the abstractions of the parse tree in graphical forms already familiar to biologists, using icons, labels, and conventions typical of publication illustrations (e.g. blocks for coding regions, line segments in a triangular "hat" over introns, ellipses at protein-binding sites, etc.). We will use code from RSVP for this purpose, perhaps even generating its intermediate language from Prolog, but the much greater power and degree of hierarchical abstraction in GENLANG will permit and require more comprehensive, global graphical conventions. These can be specified by attributes of the grammar (just as for the current parser visualization tool), and where necessary with post-processes that take the (appropriately) list-structured parse tree as their input. A crude version of the latter technique has already been used by the PI and a collaborator to generate publication-style output from a parse tree [58].

It should be emphasized that such output is not limited to standard genes or to linear depictions. Biology has given rise to many imaginative and sometimes rather abstract techniques for scientific visualization of sequence data, e.g. Nussinov plots of secondary structure [45], helical wheel plots of peptides to detect amphiphilic regions [13], and widely-used conventions for depiction of secondary-structural motifs of proteins, not to mention the more obvious uses of graphical plots of hydrophobicity [31], etc. We believe that a term-structured, hierarchical data structure such as a parse tree, and an underlying pattern-recognition system capable of expressing and detecting distant and complex dependencies, will be a superior base for developing these as well as entirely novel visualization techniques as new grammars are explored. (Note also that grammars are not limited to one or even two dimensions—see ¶3.5.3).

Finally, as visualization techniques are explored in connection with the parser, and as the limited expressive powers of RSVP are thus expanded upon, it will become desirable to reincorporate certain of the images designed for these purposes back into the X-based graphical interface, so that again they may be presented to the user in a dynamic fashion during the parse and be made available for meaningful interaction. This will be a long-term goal of the project.

3.1.2 Grammar Development Interface

All the graphics development to date, and that described in ¶3.1.1, is to do with the output of the parser. At least as important ultimately will be the creation of a graphical interface to support grammar development and management. Grammars, like any code, are intimidating and tedious to maintain in a monolithic listing, and despite the benefits of parser visualization, debugging of logic grammars using the standard four-port model of Prolog can be very challenging. Also, the management of multiple parameters and attributes, particularly involving powerful logic variables and list structures, becomes cumbersome. These issues can be addressed to a very great extent by this grammar development interface.

Grammars are inherently declarative, hierarchical, and modular. We will take advantage of this by creating a window-based grammar management system in which each major nonterminal will have its own stylized, named window that can be popped up at any time for modification. These will be under the control of a master grammar management window, with iconic representations of the nonterminals and a cross-reference graph connecting them so that it is immediately apparent what invokes what. (Note that this will be isomorphic to the derivable parse trees, modulo recursion and disjunction.) A more compact form of this will be "walking" menus to quickly descend a derivation tree. The attribute lists will be removed from the nonterminals, and instead represented as *fields* of the standard nonterminal windows. Appropriate "widgets" will be used to further simplify the representation and modification of attributes; for example, cost constraints will be entered by "slider" buttons with optional type-in.

Gaps will have a special window format that will allow much more flexible specification of costs to be associated with them. While it is now possible to specify the cost of a gap as a mathematical function of its size (in the simplest case just some multiple of its length), one might wish to more easily design a complicated discontinuous function. For example, a gap might most commonly be within a certain size range, with the distribution falling off sharply on one side and more gently on the other, perhaps with other inflections. The gap window will have a rudimentary drawing tool, by which the user can simply draw the cost function versus size. Buttons will also be able to automatically draw certain standard functions, such as a normal distribution, with touch points that the user could drag to change mean and standard deviation of gap sizes. (These techniques are now common in graphics programming, and it is likely that public domain code could be re-used in this and many other cases.)

The grammar manager will also assist in revision control, maintaining multiple versions of a given nonterminal from which the user can quickly reselect the active one. This feature we have found from our own experience to be highly desirable. For initial grammar design, one special window will be designated as a "scratchpad" available for rapid recompilation of selected features, which will be capable of dynamic modification during pauses in the parse, based on the users' observations of the parser visualization

Perhaps most importantly, we will develop a source-level debugger, so that less experienced users will be able to observe a single-stepped parse at the level of the grammar, rather than the underlying (and much more complex) Prolog code. Quintus' source-level debugger does not currently support such "term-expanded" source code, but the fact that the grammar rules all have standardized hidden parameters, and guard clauses and "trail hooks" to implement various constraint attributes, means that such a source-level debugger would be a reasonable longer-term goal. Ultimately, we would extend this support to the level of the graphical grammar manager itself.

3.2 Embeddability

As the repertoire of sequence analysis software available increases, and with the growing trend toward integrated systems, the ability of such a system to *embed* other algorithms, and to itself be

embedded, becomes increasingly important. We will address both of these needs.

3.2.1 Embedded Algorithms

The ability to embed pre-existing code is critical if for no other reason than to avoid re-implementation of what are often very sophisticated algorithms, in the public domain or otherwise available. Quintus Prolog offers a foreign function interface to 'C', Pascal, and Fortran, as well as UNIX system calls. In addition, the logic grammar paradigm encourages in-line calls to arbitrary code (Prolog or otherwise) in the bodies of grammar rules. Thus there should be no intrinsic obstacle to incorporating other well-modularized code into the pattern recognition process of an individual rule.

We distinguish between algorithms that we will actually embed in the grammar as built-ins, and mechanisms we will create to make it easy for new algorithms to be embedded by users. In the former category is dynamic programming minimum-cost alignment [60], which will be implemented as a generalization of our cost function, i.e. allowing indels as well as mismatches, where so designated by the user. While this algorithm is $O(m \cdot n)$ in both time and space for m and n the lengths of the sequences, in our paradigm there is a maximum cost threshold which will allow us both to limit the range of input examined (n) and the distance moved from the central diagonal of the cost matrix [18]. While these speedups would not be available with the non-linear gap cost functions that are preferred in these algorithms, it must be remembered that our purpose is just to augment our simple cost function for short probes and not to replicate full alignment of lengthy pairs of strings. For small enough probes and cost thresholds, in fact, a linear-time recognizer is possible, albeit with greater cost at compile-time [66]. For the full dynamic programming method, we can also save a factor on space, since no alignment need be returned, only a cost [67]; however, when such features are preceded by GENLANG gaps, it will be necessary to maintain a two-dimensional matrix in order to "scan" for an acceptable local alignment without discarding data on each increment of the starting point. We can also make use of the already-existing hash table to speed up matching on longer strings, where somewhat less sensitivity to substitutions is acceptable [69]. Finally, in the longer term, we will replace the hash table of the input with a *suffix tree*, which can also be built in linear time [40] and which can be used not only to speed up dynamic programming [41], but also to return *longest* direct repeats from any position (not just the set of those as long or longer than the k-tuple size of the hash table). We will explore the possibilities of analogous data structures for longest *inverted* repeats, and of "n differences" algorithms based on suffix trees for loose matches [41].

In embedding algorithms, our philosophy is to use the grammar as a framework to house and apply possibly expensive algorithms selectively against *regions of interest* (as they are called in the signal processing community). Not only does this achieve the economy of ignoring "boring" regions, but it allows the results of the algorithms to be assembled into a hierarchical context, and to have their results combined with those of other algorithms as well. In order to accomplish this with any particular algorithm, it must be made to fit into the GENLANG parsing paradigm, which means that (1) it must be applicable to some window on the primary sequence or, in combination with a preceding GENLANG gap, it must be able to scan some *range* of sequence presented to it as a 'C' character array; (2) it must be capable of having any parameters passed to it as arguments, which will then be controlled and thus "tunable" at the attribute level of the grammar; and (3) it must return a result in one of four forms: (a) either logical success or failure at any given position, or (b) within a range, a position at which it first succeeds, or (c) in either of the first two cases, logical success together with some scalar value representing "goodness of fit" expressed as a cost (which for many algorithms may involve reversing its sign), or (d) within a range, a *set* of positions representing logical successes with scalar cost values falling below an imposed cost threshold. (The

forms (c) and (d) are closely related to those implicit in *stochastic* grammars [21]).

We have found that a wide variety of currently-used algorithms can be made to conform to this paradigm, ranging from simple weight matrices to neural net recognizers to measures of local similarity. As an example of this approach, our tRNA grammar could be improved upon by passing putative introns to a Zuker program [71] (or similar RNA-folding algorithm) and translating the minimum energy returned into a cost. The cost threshold could be adjusted so that the quality of the entire tRNA match would depend in part on the ability of the postulated intron to form tight stems, which is observed to occur in forms too widely varied to express as a single string variable expression [34]. On the other hand, applying this $O(n^3)$ algorithm beyond the limited context of introns "proposed" by the grammar would be very slow.

For algorithms which for theoretical or practical reasons cannot be easily integrated into the grammar paradigm. it may nevertheless be desirable for a grammar to make use of their output. In many cases this can be conveniently accomplished by running a global algorithm as a preprocess and storing its results in the GENLANG *chart*; this is conceptually appropriate since such preprocesses produce intermediate results, which is the purpose of the chart vis-à-vis the grammar, and it is also practical insofar as the mechanisms already exist for the grammar to make use of the chart (even exclusively—a grammar can refer *only* to charted nonterminals, and never even have recourse to the primary sequence). As will be seen, charted data can range from fine-grained numerical properties to more typical course-grained features.

(It may be supposed that this use of preprocesses is a force-fit with respect to the grammar formalism; in reality the correspondence with theory *and* practice is quite natural. Context-sensitive grammars can be used to "side-affect" or systematically alter an input string to achieve the kind of "piped" architecture used in fields such as signal processing to successively filter and transform data streams. The PI has written extensively on the use of logic grammars for this purpose [54, 57, 59], and others have used similar "cascaded" grammars to good effect in natural language processing [70], where there is a natural progression of phonetic, syntactic, semantic, and pragmatic analyses.)

3.2.2 Embedding in Other Systems

It is our prejudice that a grammar-based system is a superior platform for marshalling a wide variety of algorithms and techniques, for applying them in a directed manner, for assembling their results into structural models that coalesce multiple lines of evidence, for rapid-prototyping and tuning parameters and cost thresholds in a uniform context, and indeed for managing sequence analysis activities at the user interface level. However, we will not impose our viewpoint on those who might wish to avail themselves of the sophisticated pattern-matching capabilities of GENLANG proper, in the context of their *own* sequence analysis platforms. Part of the proposed effort will be to make GENLANG easily embeddable in other packages.

Quintus Prolog programs are now fully embeddable in other languages, e.g. they can be called directly from 'C' code. The design of GENLANG has been such that an invocation of the parser at the top level is nothing more than an ordinary Prolog goal, capable of being called by other Prolog code or indeed by 'C' code; parameters are easily packaged as well. Quintus run-time systems can be created for established grammars so that such embedding can be done without the cost of a full development package. We will assist other computational biologists in performing such integrations (see ¶3.6).

In addition to simple arguments that may be returned by GENLANG to a governing system, some such systems may wish to make use of the structured output of the parse tree. This will depend on the ability of the outer system to pass term structure, but Prolog could be of help in flattening it if necessary. Moreover, not only is the form of the parse tree customizable under the control of grammar attributes, but being a canonical form it would also be possible to translate it to a

wide variety of flat forms using *generative* logic grammars that specify the output expected by the embedding system; the PI has used this technique in a number of instances [55, 57].

3.3 Parser Technology

At the heart of GENLANG is a fundamentally unsophisticated recursive-descent parser, which however is tremendously flexible, residing as it does in a logic programming framework, and which has been optimized for many characteristics of the domain. We will continue to enhance the parser, using more advanced techniques from the literature, and of our own invention. In this effort, we will benefit from a collaboration with Prof. Mitch Marcus of the Computer and Information Science Department, an authority on parsers for natural language (in particular, probabalistic parsing); a letter from him is attached.

3.3.1 Parser Extensions

Some of the aspects of grammars that are most daunting to newcomers, such as the heavy use of recursion, can be handled more automatically by the parser. We will introduce a notation, similar to the very intuitive *Kleene star* operator of regular expressions, that will denote iteration in the body of a grammar rule and generate optimized code for recursion of various types (e.g. longest, shortest, least cost, etc.). This will also serve to avoid non-terminating constructions, which in fact can also be addressed even in left-recursive rules through intelligent use of GENLANG's chart, by incorporating *Earley deduction* [46] or even by bottom-up parsing.

Chart parsing affords not only greatly increased efficiency on backtracking by storing intermediate results, but also a convenient persistent database of partial results to supplement the parse tree; a facility will be implemented to save the latter, for examination or re-use. The chart mechanism that we have implemented involves not only a *well-formed substring table* in Prolog, but a facility for skipping rapidly over gaps using two auxiliary bit arrays for each charted nonterminal, marking where the nonterminal has been tried and where it has succeeded. This is most suitable for features that occur at an intermediate granularity, in the range of one per 100-1000 base pairs. For less frequently occurring features, we will extend the chart parser to save the positional information in the Prolog database rather than in 'C' arrays; this will involve somewhat more computation but a great space saving. It will also be the appropriate format for importing the results of some algorithms run as preprocesses (see ¶3.2.1). For more frequently occurring features (>one per hundred bps), we will also make available a form of chart which exclusively uses 'C' arrays, with no entries in the Prolog database. This fine-granularity chart will have, in addition to positional bit arrays, dynamically-allocated integer arrays wherever necessary to store the span, cost, and user-specified parameters associated with the features; this will be the appropriate mechanism for storing local scalar properties of sequence (e.g. base class ratios, probabilities of coding regions from Markov or connectionist models, hydrophobicity of polypeptides, helix- or sheet-forming tendencies, etc. as well as arbitrary values defined by the grammar). In this case also some such values will be the result of preprocesses that will inform the higher-level grammar. The granularity of the chart and other characteristics will be specified as attributes of the corresponding nonterminals so that, as always, widely-varying sources and formats of information will be uniformly integrated into the formalism.

Certain embedded algorithms will be integral to the grammar translation and parsing process. Our current weight-matrix feature, for example, is not just a recognizer but an optimizing compiler of disjuncts of exemplars (see Progress Report) which reorders the parse to examine positions in order of information content. We will add additional attributes to this feature to allow correlations of base doublets and perhaps triplets, negative exemplars, and other variations on weight matrix methods where they can be neatly expressed as an adjunct to a formal grammar (e.g. the RTIDE

[68] and CONSENSUS [64] programs). For that matter, there is no conceptual reason other than the outrageous "compile times" involved that such grammatical formulations of exemplars could not be used with machine learning methods such as decision trees and neural networks. (Two of our collaborators, in fact, have used grammars as *domain models* in effect to "bias" connectionist learning methods in this domain [44, 61].) In any case, the operation of the parser will be kept consistent with the use of such methods in the most efficient manner possible (e.g. parse reordering), and also with ancillary data structures including the current hash table and the proposed suffix trees.

It will have been noted that the efficient operation of the parser is inextricably involved with the treatment of GENLANG gaps as search engines. The facility for passing lazy gaps will be extended to efficiently support the use of gaps with arbitrary cost functions (see ¶3.1.2). In the short term, we will alter the parser such that the size of the gap tried will automatically proceed from the least costly (most likely) to the most costly (least likely), within the current cost threshold; thus, if a gap is designated by its cost function as being normally distributed about a mean of 10, the order of gap sizes tried might be 10, 9, 11, 8, 12, 7, etc. until the cost was too great. In the longer term, we would extend this idea to recursive nonterminals and especially the combination of a gap and a nonterminal, as follows: Currently, lazy gaps allow the user to designate that the search on the string should be either breadth-first among disjuncts (using the attribute `wide`), or depth-first (`deep`), as described in the Progress Report. We have recently introduced a new attribute for best-first parse of a subtree (`best`) which will find all combinations of gap and nonterminal within the range of the gap, and then succeed with each of those combinations in order of least cost. While this will not reduce the computation time for potentially expensive nonterminals, it will often be appropriate to perform all parses on a suitably restricted subtree in order to rearrange the order in which they are presented to the user in an interactive session. Moreover, we will investigate the use of A* search or similar techniques to prune our best-first search mechanism, so as to produce a speedup even for all-parse results.

3.3.2 Island Parsing

Island parsing is bottom-up parsing in which the order is not necessarily left-to-right, and where it may in fact be "inside-out." We routinely employ such techniques in GENLANG already, using positional operators which allow for arbitrary movement on the input string, and thus accomplish a re-ordering of evaluation. For example, in the tRNA grammar described, for scanning long sequences we actually search for the relatively well-conserved loop of the TΨC-arm first, then back up and find the remainder of the molecule, for a several-fold speedup. However, this re-ordering is "manual" insofar as it is explicitly directed in the grammar. We will investigate methods by which the right hand sides of grammar rules can be automatically re-ordered by deducing the features that are likely to be most discriminating and/or deterministic, and searching for them first. This is already done in the case of weight matrices, and with abstract evaluation of rules should be possible (at least with non-recursive subtrees) as well; Staden [63] has described related techniques for certain biological patterns in particular. This will unburden the user of having to deal with this particular procedural aspect of the parser. It will require particular attention to interactions with the chart [36], but we feel that our current architecture can address these concerns.

One method we have developed for accomplishing a variety of parse re-ordering is the use of *recursive derivation*, or executing a parse within a parse [53]. This is useful in controlling multiple-level parsing (e.g. assembling exons and then parsing the polypeptides) and also for expressing the *superposition* of elements in the string (which can also be denoted using the ampersand connector to specify features that must co-occur or overlap on the input) [53]. We have shown that the combination of superposition and gaps can express a wide range of interval relationships [2, 54].

The current version of GENLANG provides limited support for recursive derivation (where it refers exclusively to predetermined spans on the already-loaded input string), and this would be extended to allow dynamic loading of additional sequence, or rearrangements of the loaded sequence.

3.3.3 Probabalistic Parsing

It is interesting to note that the field of natural language processing has recently turned to *probabilistic* parsing to address longstanding problems of building large robust grammars and performing best-first parsing [11], just as the most successful gene-finding algorithms are using empirically-derived statistical methods. We will investigate whether techniques can be borrowed from this very active field of research that will be useful in our domain, such as the methods used to gather statistical data about structured features [8, 39] and the incorporation of probabilities into chart parsers [36, 38].

Stochastic grammars are a formal system for attaching probabilities to grammar rules, such that a probability is derived for any successful parse based on multiplying the values of every rule invoked [21]. We note that our current cost system is incompatible with this, since costs range over positive reals, with zero being "most probable," and are summed over subtrees of the parse. However, it should be possible to implement other cost models using the existing hidden parameter mechanism, or even to mix them under the control of attributes. It would also be necessary (and potentially very useful) to allow for alternate models for combining evidence from disjunctions, conjunctions, and concatenations of features (concatenation being the ordinary interpretation for adjacent grammar elements); currently costs are summed in the latter two cases, and a minimum taken over disjuncts, but arguments can be made for multiplying concatenations (as in stochastic grammars), taking the maximum of conjuncts, etc. We will investigate the utility of such measures as the need arises.

3.3.4 Gene Parsing

As a means of driving the parser development in useful directions, we will further develop protein-coding eukaryotic gene grammars, incorporating the methods that have been published recently for detecting splice junctions and coding regions into our hierarchical syntactic parse. Some such methods will be straightforward to integrate, such as the filters used in GeneID [25] regarding base frequencies and covariances between base positions in adjacent codons; such heuristics can be incorporated into the current grammar's cost mechanism to contribute to decisions about the desirability of continuing an exon versus trying a weak splice donor. Other methods may depend upon planned enhancements to the parser or embedded algorithms, such as statistical/connectionist measures; for this we will depend upon the advice of our collaborators who work in these fields [44, 61]. Most importantly, the combinatoric nature of this problem will depend upon adequate implementations of the proposed best-first parsing mechanism in conjunction with the chart. We will also go beyond current algorithms, taking advantage of the grammar's strengths to incorporate contextual information (e.g. upstream signals), globally-imposed constraints (e.g. apparent periodicities in spacings between adjacent introns [4]), and recursive analyses (e.g. examining postulated exons by recursive derivation for known protein motifs, perhaps in exon-bounded functional domains, and for continuity of characteristics, such as hydrophobicity or amphipathic periodicity, even crossing intron/exon boundaries).

While the gene grammars developed may be the most taxing problem addressed by GENLANG, it is important to emphasize that GENLANG is not intended to be yet another gene-finder. Possibly it's greatest general utility will be at a level of complexity midway between motifs and eukaryotic genes, finding features with sufficient higher-order structure and variation that other pattern-matchers are deemed inadequate, yet not so complex as to require extraordinary skill at devising grammars.

This is not to say that gene grammars will be inaccessible to any but experts; once written, gene grammars should provide a perspicuous framework within which the average developer can alter cost thresholds and other parameters, and switch alternative heuristics in and out, with ease. At the same time, the expert will have available the power to integrate entirely novel sources of evidence, and the end-user will have a practical run-time system for visualizing putative genes and their variations in new sequence data (see ¶3.6).

3.3.5 Logic Programming Technologies

Given the rate at which new technologies are being introduced, both in relation to parsing and logic programming, it is not possible to predict which will be of use in the longer term to this application. However, technologies from the latter field that will be followed in particular include: (1) constraint logic programming, which in certain defined domains allows for a logic-oriented formulation of problems that can be solved very efficiently by algorithms producing *sets* of answers (i.e. parses); (2) abstract evaluation, where examination of the form of a logic program (i.e. grammar) may lead to useful predictions concerning its behavior in advance of its actual execution; and (3) deductive databases, a field much concerned with efficient evaluation of recursive queries in a logic framework.

3.4 Input Formats and Processing

The current means of conveying sequence data to the parser will be enhanced, in step with the increasing sophistication of biological sequence databases. A particular design goal will be to continue to specify such queries in a form consistent with the linguistic nature of GENLANG, and with the theme of attibutes modifying grammar objects. For these purposes, we have enlisted the collaboration of Profs. Peter Buneman and Susan Davidson of the Computer and Information Sciences Department, whose areas of research include distributed heterogeneous databases and advanced database query languages. (Our laboaratory has met with them on a biweekly basis for the past year; a letter from Prof. Buneman is attached.)

3.4.1 Flat File Databases

The current DCG-based parser for input files handles GenBank flat files in IntelliGenetics format, and is easily modified for slight variations, such as GCG format. It can also be extended to other flat file formats with relative ease, and this will be done for one-of-a-kind databases of interest, such as standard training sets, transcription factor databases [24], and the like. It will also be straightforward to extend the current directory and file-system input manager to scan subdirectories in a selective fashion, which will be useful in sequencing projects.

It will also be possible to extend the flat file input grammar to parse the features table of GenBank (or analogous structures in other databases) and enter this information directly into the chart. This will allow GENLANG to combine this information with the parse of the primary sequence, for example confining itself to regions in a certain relationship to known features. In fact, given the architecture of the chart parser it will be possible for GENLANG grammars to parse the features table exclusively, without even consulting the primary sequence. The features table parser will also supply information for portrayal on the graphical interface, at the direction of the user. It will be possible to write grammars which search for instances of feature table entries with given characteristics and/or relationships to each other, and then use the log files to save the structure and primary sequences, e.g. for machine learning applications.

3.4.2 Relational Databases

We will also adapt our "linguistic" query mechanism to work with relational databases, such as the relational form of GenBank. Just as the flat file parser uses a logic grammar parameterized by attributes expressed at the command line query, it will be possible to use an analogous *generative* logic grammar to formulate corresponding SQL queries. (We already have considerable experience in interfaces from Prolog to relational databases, in particular in adapting a Quintus Prolog interface to Sybase [23].) The functionality at the level of GENLANG will be the same (though more efficient), and in fact the user need not even be aware of the underlying mechanism of data access. We will, however, work to augment the attribute system for greater flexibility, consistent with both the flat and relational formats, e.g. to allow for multiple constraints on a single field and negation (which are not currently possible).

At this point we will also extend the semantics of the right hand side of GENLANG queries, to allow for various types of *combinations* (effectively, string-oriented joins) of entries. Implicit in current multiple-entry queries is the notion of *disjunction* among those entries: a query to `genbank` is really a query against each of its entries disjoined, e.g. `(...) ==> seq1 | seq2 | ... | seqN`. As noted in ¶3.3.2, however, other connectives are used in our grammars, including ordinary concatenation; this can be used on the right-hand side of a query, by the notation `(...) ==> seq1, seq2, ... , seqN`, to cause the entries to be examined in a connected sequence. Conjunction, on the other hand, would require a query to succeed separately in *every* entry presented to it, e.g. `(...) ==> seq1 & seq2 & ... & seqN`. This could be used to find structural commonalities fitting a given general pattern among a series of entries, e.g. in the simplest case a string variable representing actual common subsequences. In examining combinations of these connectives with gaps, we have found [54] that this simple algebra has great expressive power to represent *interval relations* [2], e.g. the superposition pattern `(seq1, ...) & (..., seq2)` allows for an *overlap* between the end of `seq1` and the beginning of `seq2`. This will also be of obvious utility in sequencing projects. As always, the rather neat declarative formulation permitted by grammars must be accompanied by efficient procedural implementations in the parser, which in many cases will go beyond the naive operational semantics already inherent in the logic programming paradigm.

3.4.3 ASN.1-Formatted Data

The National Center for Biotechnology Information has adopted ASN.1 as its data interchange format for GenBank and affiliated data [43]. ASN.1 parsers are easily implemented (by design), and we will either adapt an existing one or create a DCG with attribute parameters consistent with our flat-file parser and relational query generator.

There is widespread interest in the use of object-oriented databases in this domain, particularly as relates to genome databases [20]. Ultimately, our paradigm may be adapted to this form of data, and in general we will be alert to possibilities for integrating GENLANG with physical and genetic map data, e.g. parsing *very* high level charted data concerning map relationships of genes, gene clusters, control regions, chromatin organization, etc. We have laid at least a theoretical foundation for a linguistic formulation of mutation and rearrangement at the level of the genome [53], and in addition have experimented with karyotype description grammars which might be integrated with structural grammars [55].

3.5 Protein Sequence Parsing

The current release of GENLANG parses only DNA sequences; we will extend it to deal with protein as well, at a number of levels.

3.5.1 Primary Structure

Adapting the parser to accept sequences of the single-letter amino acid code is straightforward at one level, but the system as a whole will need to be adapted to display protein sequence in the parse visualizer, in the pretty-printed tree format, etc., and in addition the chart will require modification. The input parser will be extended to read protein sequences from GenBank features tables, including the cases of multiple gene products from single entries. The most significant change will be to the cost mechanism for string matching; PAM matrices for this purpose will be implemented in a manner consistent with the grammar formalism, as a special class of rule which is compiled directly into the string matching routines rather than the usual Prolog rule. (Such rules have already been prototyped in GENLANG for complementarity of string variables, e.g. assigning fractional costs to G-T pairs). This relatively simple series of steps will allow us to bring GENLANG to bear on the growing body of knowledge of protein motifs (*à la* Bairoch's PROSITE database); for features such as zinc fingers, for example, GENLANG will express in a natural way multiple classes sharing structural features, residues with general but not absolute rules, size variations, and incomplete or "degenerated" copies [28], and it will be especially useful as the "genetic code" of the DNA-binding regions becomes apparent.

3.5.2 Secondary Structure

Secondary structural information will also be incorporated into GENLANG grammars. The work of Rawlings has demonstrated the utility of a logic-based approach in pattern-matching of structural motifs involving secondary structural elements [49], and the *Ariadne* system has used pattern-directed inference to produce a number of biological results combining secondary structural information with primary sequence patterns in what amount to enhanced regular expressions [33]. Like these systems, GENLANG could use a preprocess to classify secondary structural regions, e.g. the well-known Chou-Fasman algorithm [10]; we note, however, that this essentially rule-based system could be implemented in GENLANG (and indeed portions of it have been [M. Noordewier, personal communication]), and variations on this and related algorithms could be experimented with in a grammatical context. By the same token, connectionist methods that have addressed this problem with moderately improved results, including one developed by a collaborator [37], could be embedded. We have argued [56] that grammars will prove intrinsically superior to regular-expression pattern-matching in this domain because of the long range dependencies among residues in apposition, as well as parallel and anti-parallel interactions in β-sheets, amphipathic periodicities in α-helices, and other inherently "grammatical" phenomena.

3.5.3 Tertiary Structure

While tertiary structure *per se* will not be addressed in depth in the proposed work, we note that the field of syntactic pattern recognition has produced grammar formalisms that describe two- and three-dimensional structures, in particular those which trace a path through space [21]. In particular, *coordinate grammars* have coordinates in space associated with nonterminals, and rules entail not only the usual rewriting but also the application of an associated function to derive new sets of coordinates [50]. As a long-range goal we would prototype a grammar-based system for describing known tertiary structures, allowing variations, substitutions of domains, etc.

3.6 Applications

To provide additional technology "pull" on all of the development work proposed above, we have arranged a series of collaborations involving both computational biologists who wish to use GENLANG

in their work, and biologists who will provide "real world" problems to be addressed by GENLANG. These collaborators span a range of three classes of users we envision for GENLANG: *(1) expert users,* computational biologists with some academic background in grammars and parsers, who will use GENLANG as a platform for experimental development of sophisticated systems such as gene-finders; *(2) developers* of intermediate-level grammars for motifs and higher-order structures beyond the ken of simple pattern-matchers; and *(3) end-users* who will not write grammars but who will use run-time systems developed in GENLANG, and who will have the ability to manipulate the graphical interface and change parameters. For expert users, our goal is to provide a versatile platform for experimentation with algorithms and heuristics for sequence analysis, with well-integrated visualization tools, support for rapid prototyping and code integration, a natural hierarchical abstraction, and strong formal underpinnings for analysis of those techniques. For developers, our goal is to create a language clear enough, a development environment helpful enough, and a parser forgiving enough, that intermediate level grammars can be written by non-experts including computational support staff and adventuresome biologists, without undue concern for efficiency and other niceties of algorithm design. For end-users, our goal is to present a run-time system with an attractive and functional graphical interface to a powerful pattern-matching system using a library of functions devised by others, with easily adjustable parameters whose effects are visually apparent, and outputs ranging from intuitive depictions of parse trees, to parse-driven visualization tools, to publication-quality PostScript specifications.

Investigators who have either agreed to establish "sites" for GENLANG to be installed immediately for use in collaboration with the PI and in their own research, or who will provide the PI with relevant biological problems, include:

- *Dr. George Michaels* of the National Institutes of Health, Division of Computer Resources and Technology. Dr. Michaels sponsors a "collaboratory" of a number of computational biologists who contribute to a suite of largely logic-based sequence and map analysis software, into which GENLANG would be incorporated. Dr. Michaels would also collaborate in applying GENLANG to specific biological problems, such as identifying RNAs sharing structure with portions of the tRNA molecule, whose GENLANG grammar he has used extensively.
- *Prof. Mick Noordewier* of the Computer Science Department and Waksman Institue for Microbiology of Rutgers. Dr. Noordewier is a long-time user of GENLANG [58] who will employ it to collect exemplars for machine learning and as a framework for the application of neural net recognizers [44].
- *Dr. Pat Gillevet* of Harvard University, Director of the Human Genome Lab under Dr. Walter Gilbert. Dr. Gillevat will employ GENLANG and related visualization technology in their laboratory, and will incorporate GENLANG into their sequence analysis platform, GDE.
- *Dr. Charles Lawrence* of the Baylor College of Medicine, Director of the Genome Center Informatics Core there. Dr. Lawrence is interested in using GENLANG for gene recognition, and in embedding GENLANG into their environment for support of large-scale sequencing.
- *Prof. Jude Shavlik* of the Computer Science Department, University of Wisconsin, Madison. Dr. Shavlik will collaborate on using GENLANG for domain theories to underlie connectionist techniques he has pioneered [37, 61], and will also make use of GENLANG to further his computational support of Dr. Fred Blattner's *E. coli* sequencing effort at Madison.
- *Prof. Steve Liebhaber* of the Howard Hughes Medical Institute at the University of Pennsylvania School of Medicine. Prof. Liebhaber, who is investigating certain complex zinc-finger motifs and secondary structural regulation of alternative splicing, will interact with the PI in creating GENLANG specifications of these systems and searching for similar patterns.
- *Prof. Stephen Mount* of the Department of Biological Sciences at Columbia University. Prof. Mount has detailed rule-based models of splice signals, showing apparent species specificity; we will collaborate with him in expressing these in grammar form and iteratively refining

through exhaustive searches.

Letters from these individuals are found in an appendix. In addition, the PI has had a number of discussions with colleagues at Penn and in the Chromosome 22 Genome Center who are interested in supplying additional biological problems for test purposes.

3.7 Distribution

GENLANG will be freely distributed to academic, government, and non-profit research institutions. The PI, who has six years of industry experience in directing software development, will manage the continued development, distribution, and support of the software in such a way as to balance the need for continued research with that for deployment, within the limited budget proposed.

3.7.1 Software Engineering Practices

As GENLANG evolves and increases its user base, more formalized software engineering practices will be adopted, of necessity, than have been used in the early development phases. Standard practices for configuration control and revision control will be followed, including extensive regression testing. Code documentation, which is currently spotty, will be formalized in anticipation of modification by expert collaborators, and in particular will incorporate modularization with change histories, etc., at a "functional block" level (typically, several Prolog relations). Also, advantage will be taken of the Quintus module system to isolate the system from possible name clashes, etc.

Compile-time checking, which currently has a repertoire of several dozen messages, will be greatly extended. Application-specific exception-handling, currently done only at a very high level for undefined grammar rules, etc., will be "pushed down" for greater effectiveness of runtime error reporting. See ¶3.1.2 for a discussion of plans for a source-level debugger.

3.7.2 Software Distribution and Support

Inevitably, the language will continue to evolve rapidly even in this period, but to the extent possible we will maintain backward-compatability and will provide assistance, both in documentation and directly, in upgrading grammars. It is anticipated that major releases (i.e. significant changes in functionality) will occur on the average of one per year during the period of this grant. "Dot" releases (i.e. minor changes and bug fixes) and patches will be added as needed. These will be accompanied by updated documentation, including release notes. The current manual contains an extensive tutorial, but not a formal language definition and compact reference manual; these will be added. The software will be supported on an informal basis, through electronic mail, to the extent possible. This will include assistance in grammar development, with priority given to collaborators. If necessary, a formal bug report system with prioritization will be adopted.

For new users without Prolog licenses or access to Sun workstations, we will offer the option of remote execution on a SPARCstation 10 "parse server" proposed under this application (also to be used by one of the staff for terminal service). This high performance machine should be able to support a number of GENLANG jobs running interactively, and will also be used for overnight batch execution of parses against databases. Remote access will include the ability to run the graphical interface, by 'X-hosting' to a remote terminal which may be a Macintosh or PC (using software proposed under the grant). We have successfully run GENLANG in this mode over the Internet from Philadelphia to both Bethesda and Berkeley, with barely perceptible delays in screen response even for parses with complex graphical interactions.

3.7.3 Grammar Repository

We will establish a repository for grammars and grammar fragments that will be a resource to users, accessible over the Internet through anonymous FTP. This will include "vanilla" gene grammars of various types available for modification by individual experimentalists, library routines of oft-used sub-grammars, grammar "tricks" such as special recursive techniques, standard cost matrices, well-known motifs in grammar form, and non-grammar-based algorithms adapted to be embedded in GENLANG. Headers will contain information on authorship, references, parameter information, and comments, and a subset of the repository will constitute a supported library for which the staff will answer user inquiries. Users and especially collaborators will be strongly encouraged to share their techniques and their experience through the medium of the repository.

The nearer GENLANG grammars approach the "linguistic" ideal of abstracted, declarative specifications, the more useful the repository will be as a knowledge base of higher-order biological patterns, independent even of the particular procedural interpretation offered by the GENLANG parser. The proliferation of special-purpose "boutique" databases of biological sequence information, and many other signs, suggest that the time is ripe for a transition from databases to knowledge bases in this field. However, the notion of standards for truly declarative yet sufficiently powerful knowledge bases is still controversial, and we believe that grammars constitute an attractive first step towards a formally well-founded descriptive system for higher-order features of biological sequences.

END

DATE
FILMED
4/9/93