

QGB: A System for Querying Sequence Database Fields and Features

G. Christian Overton*, Jeffrey S. Aaronson, Juergen Haas, Julie Adams

Department of Genetics
University of Pennsylvania School of Medicine
Room 475, Clinical Research Building
422 Curie Boulevard
Philadelphia, PA 19104-6145

Internet: coverton@cbil.humgen.upenn.edu

Running Head: Sequence Database Query Language

Keywords: query language, logic grammar, parsers, biosequence database

Abstract

We have developed a general system, QGB, for performing complex queries on the information in the DDBJ/EMBL/GenBank databases, including queries over the structural features of sequences implied in the FEATURE TABLE. Queries are formed in an SQL-like syntax with language extensions to support complex types (e.g., sets, ordered sets and records) appropriate for representing and querying sequence data. A novel aspect of QGB is its ability to deduce missing features and infer relationships among features as a consequence of constructing a parse tree of sequence structure from information described in the FEATURE TABLE. The grammar for the parse tree is implemented in a customized form of the Definite Clause Grammar syntax of the logic programming language Prolog. The logic grammar formalism was chosen because it provides a perspicuous representation for features and constraints, and Prolog provides an execution model for the grammar rules. Construction of the parse tree also identifies inconsistencies and errors in the FEATURE TABLE which can in some cases be automatically corrected and used to generate an augmented version of the table.

1 Introduction

The DDBJ/EMBL/GenBank (DEG) nucleic acid sequence databases contain, in addition to the sequence data itself, considerable ancillary information including the type of sequence, a free-text description of the sequence, the organism from which the sequence was derived, references in which the sequence is described, and a list of biologically significant features, such as exons, introns and coding regions, corresponding to subsequences of the sequence. The latter information is provided in the FEATURE TABLE of an entry using a rich definition language that specifies a subsequence feature type and location along with structured comments (DEG Staff, 1992).

A number of tools have been developed for efficiently extracting individual sequences from the

database and performing rapid sequence similarity searches, but systems for carrying out general queries on the ancillary information, especially the sequence structures implied in the FEATURE TABLE, have been less successful. Nonetheless, the demand for such systems has continued to grow as researchers seek to use the databases for more complex tasks that require, for example, retrieving sequences based on their characteristics (e.g., all invertebrate genes having an exon containing only 3'untranslated region) or extracting subsequences based on their structures (e.g., donor-acceptor splice junction pairs, proximal promoter regions, or contiguous coding sequences). Some significant progress towards more powerful query and retrieval systems has been made by several groups and includes a relational database implemented by the GenBank group at Los Alamos National Laboratory (Cinkosky et al., 1987, Burks et al., 1992); SRS, a custom query language developed at EMBL (Etzold, 1992); and Entrez, an information retrieval tool developed at NCBI as a companion to the ASN.1 formatted version of the nucleic acid and protein sequence databases (NCBI Staff, 1992).

Each of these efforts satisfies some of the capabilities desired in a general query system: The GenBank relational database, implemented in Sybase, is able to formulate and execute complex queries like "retrieve all hemoglobin gene sequences where the organism is a primate" using the relational query language SQL (Structured Query Language). SQL, however, can not perform other reasonable queries such as "retrieve all first 5'splice junctions from primate hemoglobin genes," which have instead required special purpose programs designed to extract a particular class of subsequences. Information retrieval tools like Entrez are even less well suited for these types of queries. SRS stands alone in having well developed facilities for extracting subsequences determined by the explicit information in the FEATURE TABLE, but even SRS misses the relatively large class of information that is implicit and must be inferred by analysis of relationships among features (e.g., introns when only exons are specified).

To address these problems, we have developed a general system, QGB, for querying all information in the DEG databases in an SQL-like language with extensions to handle complex data types (sets, multisets, and lists) useful for representing sequence structure information. Within the QGB framework, we have implemented an interpreter for the FEATURE TABLE that deduces missing features and infers structural relationships among features as part of the process of building a parse tree of sequence structure that is subsequently used as the primary data object in queries on sequence structure (Overton et al., 1989). The grammar and parser for sequence structure is implemented in the Definite Clause Grammar syntax of Prolog, a logic grammar formalism. This approach, based on the pioneering work of Searls (Searls, 1988, Searls, 1989, Searls, 1992), has the advantage that grammar rules provide a perspicuous representation for features and constraints, and Prolog provides an execution model for the rules. The parse tree for each entry can be queried on the fly or recorded in a database and queried later without the need for re-parsing the entry. The current version of QGB operates only on flat files in GenBank format, but it can easily be altered to cover other flat-file formats or used as a front end to the NCBI files in ASN.1 format, or relational or object-oriented databases.

It is not a trivial task to infer implicit information in the FEATURE TABLE: Aside from outright errors that are inevitable in any large, complex database, FEATURE TABLE information also suffers from ambiguous and improper use of the description language (e.g., mRNA variously used to mean primary transcript, mRNA or exon), and relegation of significant information (e.g., gene names in multi-gene entries) to free-text fields in comment fields. Presumably, most of these errors and inconsistencies arise because investigators are encouraged to directly submit annotated information to the database. The relatively few errors introduced into the database by these untrained annotators are a small price to pay to keep the database current, so it is important to develop tools to detect

and correct such errors without inhibiting investigators from submitting annotation. QGB forms the basis for this type of tool; grammar rules can be viewed as encoding constraints over the features, and logical, syntactic and usage errors can be detected as a consequence of the failure to generate a valid parse. Many of the ambiguities in representation and some errors in the data can be automatically corrected by analysis of features during parsing and in the parse tree. Therefore, QGB can be used to scan database entries for errors and subsequently produce a corrected, augmented version of the FEATURE TABLE.

In this paper, we describe the major components of QGB: the Flat-File Parser which translates files in GenBank flat-file format into a Prolog database, the QGB query language, and the Sequence Structure Parser which builds a parse tree of sequence features by analysis of the FEATURE TABLE. We begin with an overview of the system and conclude with a discussion of the current capabilities and future development of QGB.

2 System and methods

QGB is implemented in Quintus Prolog version 3.1 and was developed and tested on Sun SPARC-Stations under SunOS version 4.1.3 of the Unix operating system. It should run with only minor modification on any platform that supports Quintus 3.1. We have, however, made extensive use of several Quintus Prolog specific language features, including the module system to encapsulate code, foreign function calls to access C subroutines, and the term expansion facility to develop Definite Clause Grammar rules, that would complicate, but not rule out, conversion of QGB to other Prologs. QGB has been ported to the highly Quintus-compatible Sicstus Prolog, an inexpensive Prolog that supports a runtime generator, as a step preliminary to creating distribution version of the system.

[INSERT FIGURE 1 ABOUT HERE]

3 Overview

Figure 1 outlines the basic steps in parsing and querying nucleic acid sequences in GenBank flat-file format. In the first step, the user poses a query in an SQL-like syntax of the form:

```
SELECT <list of fields> FROM <list of sequence files> TO <results file>
WHERE <list of conditions>
```

where tuples of the <list of fields> are returned in the <results file> from entries in the <list of sequence files> when the <list of conditions> is satisfied. QGB processes the query by sequentially parsing each entry in each file into its logical components using the Flat-File Parser (FFP) module. During this step, QGB checks to see if any conditions in <list of conditions>, fail in which case further processing of the entry is skipped. If a subsequence is requested, the Sequence Structure Parser (SSP) is invoked to build a parse tree of the entry's sequence structure by examination of information in the FEATURE TABLE. The parse tree is then passed to the sequence query handler which extracts the set of requested subsequences from the sequence data.

The major components of QGB are written either in standard Quintus Prolog or in the Definite Clause Grammar (DCG) syntax of Prolog, although in the case of the SSP, the DCG has been highly customized. The DCG formalism was developed in parallel with the Prolog programming language as a declarative representation of the syntax of natural and artificial languages. Since DCG rules can contain embedded Prolog calls and further since DCG rules are converted to Prolog when interpreted, DCGs have the full power of the Prolog programming language available on the one hand while maintaining the conceptual simplicity of a grammar formalism on the other. This combination supports the rapid development and easy maintenance of the parser components of the system.

[INSERT FIGURE 2 ABOUT HERE]

3.1 Sequence Structure Parser

Investigators are often interested in extracting subsequences (e.g., coding regions) rather than the complete sequence from an entry. In principle, it should be possible to determine mechanically where the subsequences lie within the full sequence through examination of information in the FEATURE TABLE (e.g., Figure 2). In practice, this is often difficult because information about relationships between features (e.g., a 5'splice junction is the overlap between an exon and an intron) and even the existence of a feature (e.g., introns when only exons are listed) is implicit in the FEATURE TABLE and must therefore be inferred. We view the inference process as one of parsing where the features from the FEATURE TABLE are presented as input to a parser (SSP), and the output is a parse tree (Figure 3) representing the hierarchical structure of the gene with missing features filled in and relationships among features explicitly represented. The parse tree then becomes the primary data object for extracting and analyzing subsequences from an entry.

The SSP is implemented as rules in a modified form of the DCG formalism. Informally, grammar rules specify the permissible arrangements, that is the syntax or structure, of grammatical elements in a domain. Usually grammatical elements can be divided into **non-terminals**, elements that do not appear on the input and are interior nodes of a parse tree, and **terminals**, corresponding to the leaves of a parse tree and the input to a parser. However, the distinction between terminals and non-terminals is blurred for nucleic acid sequence (NA) grammars: As shown in the parse tree of Figure 3, subsequences which may be considered as “terminals” in one context (e.g., exons as subsequences of an mRNA) may become “non-terminals” in another (e.g., coding sequence as a subsequences of exons). Given a set of features corresponding to nodes from different levels in a

parse tree, our goal is to construct a complete parse tree.

DCG rules are of the form `LHS => RHS`, where the `LHS` (left-hand side) contains at least one non-terminal and the `RHS` (right-hand side) any combination of terminals and nonterminals. The `LHS` non-terminal in the toplevel grammar rule is termed the start symbol. The simplest form of a NA grammar rule is:

$$\text{transcription_unit} \Rightarrow 5'\text{flank}, \text{primary_transcript}, 3'\text{flank}. \quad (1)$$

where `transcription_unit` is a primary transcript plus an indeterminate amount of upstream and downstream sequence.

[INSERT FIGURE 3 ABOUT HERE]

As we have previously noted (Overton et al., 1989), NA grammatical elements are conceptually equivalent to intervals of NA sequences (i.e., subsequences) and formalisms developed for reasoning about temporal intervals (Allen, 1983) can be extended to cover NA intervals. In this context, a natural interpretation of grammar rules is as interval relationships where ‘=>’ means the interval on the `LHS` contains the intervals on the `RHS`, and a ‘,’ between intervals means that the end of the first interval is the beginning of the second interval. This is consistent with the usual **dominance** (part-whole) and **precedence** (order of parts) relationships implicit in grammar rules. Other interval relationships such as overlaps, starts, and ends can also be modeled within a grammar rule formalism (Pastor et al., 1991).

Grammatical elements (features) on the input list are ordered by their start positions, lengths and ranks in the grammar hierarchy; for example, an exon occurs before a CDS fragment with the same boundaries. When expressions in square brackets, `[]`, appear on the `RHS` of a rule, they remove grammatical elements from the input list, and when they appear on the `LHS` they place elements back on the input list. Therefore, an element can be removed, examined and replaced on the input

list as in the following example which tests if the 5'flank boundary has been reached:

```
5'flank, [primary_transcript(S,E,I)]=> (2)
    gap, [primary_transcript(S,E,I)].
```

where the logic variables `S` and `E` represent the start and end positions of the interval, and `I` provides context information about the features.

Grammar rules can also express alternative expansions of a LHS (disjunction):

```
5'flank => promoter, 5'flank. (3)
5'flank => gap, promoter, 5'flank.
```

and recursion:

```
primary_transcript => exon, intron, primary_transcript. (4)
primary_transcript => exon.
```

Practical grammars also need escapes to Prolog, indicated by curly brackets `{}`, to handle exceptional situations such as erroneous and missing input data:

```
primary_transcript(S,E,I_primary_transcript) => (5)
    [primary_transcript(S,E,I_primary_transcript), mRNA(S,E,I_mRNA)],
    io_list(InList,OutList),
    {resolve_mRNA_with_exons_and_cds(InList,S,E,
                                     I_primary_transcript,
                                     I_mRNA,OutList)},
    primary_transcript1.
```

where the logic variables `S` and `E` are again the start and end positions of the intervals, `InList` and `OutList` are the input and output lists, and `I_primary_transcript` and `I_mRNA` provide context information for their respective features.

Rule 5 illustrates a case where the parser must reconcile information about the primary transcript and the mRNA with information about exons and coding sequences: The SSP examines the input list, `InList`, and possibly modifies it in the predicate `resolve_mRNA_with_exons_and_cds` before

passing it along as the output list, `OutList`, to the next stage of the grammar. The predicate, `io_list`, is provided as a hook to view the input and output lists which are otherwise hidden within the grammar rules.

Using rules similar to the above, we developed a grammar tailored for eukaryotic protein coding genes (Overton et al., 1993). Figure 4A shows an example of a parse tree generated for the human α -hemoglobin gene (`HUMAGL1`) from the `FEATURE TABLE` in Figure 2. Note that the `SSP` has inferred that precursor RNA should be labeled `primary transcript`, the two features labeled `mRNA` are actually exons, and that there are two introns missing from the original `FEATURE TABLE`.

When new features or modifications to existing features are inferred in the database (even obvious ones like a missing intron), it is essential to annotate the evidence that supports the feature. A complete record of the inference chain used to deduce the feature, such as that used in the truth maintenance systems developed for knowledge representation (McAllister and McDermott, 1988), are too expensive in terms of storage requirements and computational overhead to be practical in this setting. Instead, we use a greatly abbreviated annotation method where each feature is tagged with a list of justifications (Pastor et al., 1991) of the form:

```
...
transcription_unit(pos(<,[0,0]), pos(>,[1708,1708]), ... ,
                  [justification(sspv1,derived,date(9,12,92), ... ), ... ]), (
...

```

where square brackets `[]`, in this case mean a list of terms. In this abbreviated view of a justification, the first argument denotes the version of the parser (or other algorithm) used in the analysis, the second the type of justification — generally `derived`, `experimental` or `unknown` — and the third the date of the justification. However, the exact structure of a justification varies depending on its type. (Note that to keep the output brief, the justification field was not shown in Figure 4A.)

Peculiarities in the FEATURE TABLE make implementing broad coverage grammars challenging. Perhaps the most difficult problems arise in entries with multiple genes or alternative versions of genes (e.g., alternative splice sites, transcription start sites, and polyA additions sites) where names for individual genes are missing or recorded in inappropriate qualifier fields; names are not enforced with respect to a controlled vocabulary; features are not consistently labeled with the gene of which they are part; and feature types are used inappropriately (e.g., mRNA variously used to mean primary transcript, mRNA or exon). The SSP is often forced to apply roundabout methods to resolve at least some of these ambiguities. For example, information for gene names is often buried in the free text of one of several different qualifier fields and the SSP performs a very limited form of text analysis to identify the names. Future versions of the SSP could easily incorporate a more powerful text understanding component to extract more of the available information.

Finally, the parse tree can be mapped into a corrected, augmented version of the FEATURE TABLE (Figure 4B) that requires only a few additions to the feature and qualifier types. The augmented table provides a consistent, uniform representation of features across all entries and further, can be used to directly regenerate the parse tree without recourse to the SSP thus significantly reducing query response times.

3.2 Flat-File Parser

The FFP provides the overall control for executing queries during the process of parsing entries into their logical components. In the first step, information from each entry field is translated into an appropriate Prolog data structure, and the query conditions are checked during parsing. The query conditions are organized so that failure of an entry to satisfy a condition is detected as early as possible. When failure does occur, the parser skips to the next entry without finishing the failed entry, significantly improving performance of the system for queries conditioned on the first five GenBank fields (LOCUS, DEFINITION, ACCESSION, KEYWORDS, and ORGANISM). Error

messages, warnings and information on unused or un-parsable features are sent to the ‘log’ file.

The FFP is written in the standard DCG syntax, making it easy to maintain and modify, and develop parsers for alternative DEG formats and other databases such as PIR. An idealized version of the top level rule for the FFP is:

```
gb_entry --> locus,test1,definition,test2,          (6)
            keyword,test3,organism,...,sequence.
```

The FFP will also optionally analyze old-style EMBL features stored in the COMMENT field. In this case, the FFP will interpret and then merge these features with the new-style GenBank features under three levels of user control: 1) ignore EMBL features; 2) translate only unambiguous EMBL features; or 3) guess at the meaning of the EMBL feature by examining the comments associated with the feature. When merging features, if no comparable GenBank feature exists then the EMBL feature is simply added to the list of features in GenBank format and a justification for the feature is recorded; if an equivalent GenBank feature exists then only the justification is updated; and if there is a conflict between the EMBL and GenBank feature, then a set of rules is applied to deduce the form of the feature that is most consistent with both features, the resolved feature replaces the GenBank feature, the justification is updated and a message is sent to the ‘log’ file.

3.3 Query Language

A QGB query corresponding to “return the locus ID, the description line, and 10 bp 5’ and 20 bp 3’ to the 5’splice junction for all splice sites in all non-mammalian genes with complete coding sequences” is (Query 1):

```
SELECT locus.id, definition, 5'splice_site = JUNC(-10,pt:exon,pt:intron,20)
      FROM '/databases/gbrel73/*.seq',
      TO myresults,
      WHERE organism =\= mammalia AND
            definition AMONG ("complete CDS" OR "complete coding sequence").
```

The query is interpreted from within a Prolog read loop, converted to a Prolog/DCG rule similar to Rule 6, and then executed. Arguments of the **SELECT** statement are returned in the file specified by the **T0** statement with the extension ‘out’ added (e.g., ‘myresults.out’) and the file ‘myresults.log’ becomes the log file. The input files designated by the **FROM** statement can be a single file or a list of files in square brackets, i.e., [file1,file2,file3,...,fileN], where the file names can be ambiguous (e.g., ‘*.seq’ means all files in a directory with the ‘seq’ extension). Arguments to the **WHERE** statement are conditions that must be satisfied for an entry before values are returned.

[INSERT TABLE I ABOUT HERE]

Conceptually, all queries are performed on a single object — ‘entry’ — which is roughly equivalent to a “universal user view” in a relational model, except that values returned by the **SELECT** operator and values tested by the **WHERE** operator can be terms, lists and sets as well as atomic types. For example, the value of ‘keywords’ is a list of Prolog atoms. Table I shows some of the valid operators in QGB and and Table II lists some of the fields and subfields, and their data types. Subfields are referenced using the dot notation as in locus.id, where id is subfield of locus containing the LOCUS ID. Sets of elements of sets or trees are extracted using a colon as in pt:exon. Elements of trees include both internal nodes and leaves. If the **SELECT** statement contains only the term ‘entry’ then complete entries in GenBank flat-file format are returned. Otherwise, the arguments to the **SELECT** statement can specify any valid field or subfield.

[INSERT TABLE II ABOUT HERE]

Arguments to **SELECT** and conditions in **WHERE** can be functions which extract subsequences using FEATURE TABLE information directly or after construction of the canonical sequence structure parse tree. The latter is greatly preferred since information in the parse tree is generally an extension of that in the FEATURE TABLE. Queries to extract subsequences can access feature types (symbolically labeled sequences) represented in the parse tree (primary_transcript, exon, intron, etc.),

subsequences of named sequences, spans across the junctions of named intervals, named concatenations of sequences (e.g., mRNA is a concatenation of exons, complete_CDS is a concatenation of CDSs), and user specified concatenations of named intervals, (e.g., `CONCAT(5' flank, 3' flank)`).

Some of these functions are shown in Table I. In addition, functions are included for extracting subsequences using numerical ranges specified in more-or-less standard FEATURE TABLE syntax, i.e., `range(JOIN(50..100,200..300,400..500))`, and indexed ranges of named features (`ALL(pt:exon)`, `NTH(3,pt:intron)`, `LAST(f:cds)`, etc). In general, these functions will return sets of feature records, not just a single feature record, and are therefore strictly beyond the capabilities of standard SQL. Union and intersection operators are also provided so that a single query can combine several sequence specifications:

```
ALL(pt:exon) \ / ALL(ft:exon)
```

which would return the union of the exons found in the parse tree with the ones listed in the feature table. (These two will differ when the parse tree contains inferred features not listed in the FEATURE TABLE, and the FEATURE TABLE contains features that could not be parsed.) Other operators available in the WHERE statement are again based on the analogous ones in SQL (Table I). These include tests of arithmetic relationships, string comparison, and substring searches over text strings and lists of text strings.

3.4 Output and Report Generation

The standard output format for query results is modeled after that typical of an SQL query: Each “tuple” is on a separate line with the attributes in order of their appearance in the SELECT statement. Some enhancement to the relational syntax is required to accommodate the complex data types in QGB. For example, the results of Query 1 are of the form:

locus.id	definition	5'splice_site
atom	string	set of span records {(integer integer sequence)...}
HUMAGL1	"Human alpha-globin ..."	{(221 251 cctggagag...)}...

where curly brackets denotes a set and elements in parentheses are records. Results of queries can also be directed to output files in user specified formats, a step equivalent to report generation in relational DBMS, or passed to other Prolog or C programs for further analysis. A reasonable knowledge of Prolog is required to customize the output format for report generation.

4 Results

We measured the performance of QGB with respect to its execution time and accuracy for typical queries. A query to extract all entries in GenBank (Release 73, approximately 260MB in size) with the term 'globin' in the keyword list or definition line (Query 2):

```
SELECT entry
FROM 'gb*.seq'
TO gb73allglobins
WHERE keywords AMONG "globin" OR
      definition AMONG "globin".
```

took 42 minutes on a SPARCstation 2. When we added indexing on the start position of each entry, the time dropped to less than 30 minutes of CPU time. (Indexing lets the FFP jump to the offset where an entry begins rather than having to read through the file to reach the start of the following entry.)

We know that the SSP is not yet capable of correctly interpreting the FEATURE TABLE information for all entries, so we examined in detail how well it performed on the subset of entries gathered in Query 2, specifically, how many correct parse trees did it produce. There were a total of 723 entries that satisfied Query 2. Of these, 15 had syntax errors in the features, 153 were RNAs and 121 had no features listed. Of the remaining 434 entries, the SSP was able to generate a correct parse tree

for 324 (75%) and a partial parse tree for an additional 32 (7%).

The SSP could not parse 78 entries (25%) listed as double-stranded DNA at all. These fell into three categories: 66 had features but none were parsable (i.e., no exons, introns, primary transcript, etc.) either because they were not genes (i.e., Alu or retroviral sequences) or because they had not been fully annotated; eleven belonged to gene segments or fragments, that is they were one of several entries which together will make up a complete gene when gaps in the sequence are filled (e.g., HUMAGL1 - HUMAGL6); and one (RABBGL1) was a pathological case where the feature identifier mRNA was used when `prim_transcript` was meant and furthermore the exons were incompatible with the specified primary transcript boundaries.

The cases where only partial parse trees could be generated occurred primarily because of usage errors (e.g., in HUMGAMGLOA and HUMGAMGLOB polyA_sig's are labeled as polyA_site's) and inconsistencies or errors in the feature boundaries (e.g., RABBGLO, SHPBBGLOB). The other cases failed because we do not yet handle foreign calls (i.e., calls to sequences in a different entry), complementary sequences, or segmented genes.

As another measure of performance, we examined how well QGB did in discovering features not explicitly listed in the FEATURE TABLE but rather inferred by the SSP. A 'globin' data set prepared from GenBank release 74 contained a total of 437 introns and 665 exons. QGB inferred an additional 130 introns (30% more) and 267 exons (40% more) not listed in the FEATURE TABLE. These are features that would not be recognized or reported by the other sequence extraction tools currently available.

5 Discussion

QGB serves both as a system for performing general queries over all information in the DEG databases, including sequence structure information, and as a tool for detecting errors in the FEATURE TABLE field. As a query language, QGB provides most of the facilities available in standard database languages while addressing inadequacies these languages have in accessing and manipulating subsequences. The query language is compatible with the work done by Buneman and collaborators on structural recursion as a query language (Breazu-Tannen et al., 1991) and future developments of the language will be coordinated with their efforts. It should be noted that the query language is largely independent of the form of the underlying database, and other versions of QGB could equally well provide front ends to flat-file formats different from GenBank, relational or object-oriented databases, or the ASN.1 database distributed by NCBI: Along these lines, we are currently implementing a QGB interface to a relational database version of the DEG database under development in our group (Hart et al., 1993). Moreover, QGB can be readily adapted as a front end to entirely different biosequence databases such as PIR.

QGB can detect some classes of logical, syntactic and usage errors in the FEATURE TABLE with very high accuracy as a result of building the sequence structure parse tree. Generally, when QGB fails to parse an entry, there is either an outright error in the FEATURE TABLE or poor usage of the description language. In either case, QGB can automatically correct many of these errors. Since we have also shown that QGB can deduce a large number of features that have not been explicitly annotated, an obvious function for QGB is as the core of a system for systematically correcting and augmenting features in existing FEATURE TABLEs and as new features are entered. The latter is especially important because it partly relieves investigators who are annotating features from understanding the full complexity of the FEATURE TABLE description language. As a pilot project,

we have constructed a corrected, augmented database for the extended hemoglobin gene family; this database also contains considerably greater detail on the hemoglobin genes — particularly with regard to transcription factor binding sites — than is available in DEG.

A major strength of QGB lies in its ability to access the canonical sequence structure parse tree generated by the SSP. Consequently, significant improvements in the coverage and accuracy of queries will come largely in the form of more robust grammars and parsers that overcome current limitations in handling such areas as foreign references, segmented entries, and some important biological features — alternative splice sites, alternative transcription initiation sites and overlapping genes. In addition, grammar rules can also incorporate analysis steps ranging from simple logical checks such as confirming the accuracy of coding sequence features to natural language analysis of free-text fields in FEATURE TABLE qualifiers. In fact, the present version of QGB already performs a very limited form of text analysis while trying to determine the names of genes in multi-gene entries and these facilities can be readily expanded. However, the single largest performance improvement will result from the implementation of specialized grammars tailored for specific organisms and classes of genes to supplement the single set of grammar rules developed for eukaryotic protein coding genes now in use.

For many users, a query response time of 30 minutes, such as that for Query 1, is not unreasonable. Nonetheless, we can substantially improve on this either by converting QGB to a relational database front end, as discussed above, or by providing additional indexing on fields and subfields in the flat files. J. Tisdall in our group has shown that indexing on the text in the LOCUS, DEFINITION, ACCESSION, KEYWORDS, and ORGANISM fields can cut the time for typical queries down to approximately five minutes at the cost of an additional 20 megabytes of disk storage for the index files (unpublished results). Future versions of QGB will therefore include index files as an optional

feature.

Acknowledgment: We thank Peter Buneman and Susan Davidson for extensive discussions on database and programming languages. This work was supported by NIH RO1 RR04026 from the National Center for Research Resources.

References

- Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.
- Breazu-Tannen, V., Buneman, O. P., and Naqvi, S. (1991). Structural recursion as a query language. In *Proceedings of the 3rd International Workshop on Database Programming Languages, Naphlion, Greece*. Springer-Verlag LNCS. To appear.
- Burks, C., Cinkosky, M., Fischer, W., Gilna, P., , Hayden, J., Keen, G., Kelly, M., Kristofferson, D., and Lawrence, J. (1992). Genbank. *Nucleic Acids Res*, 20 Supplement:2065–9.
- Cinkosky, M. J., Fickett, J., Nelson, D., and Marr, T. G. (1987). The restructuring of GenBank. Technical report, Los Alamos National Laboratory, Los Alamos, NM, USA.
- DEG Staff (1992). The DDBJ/EMBL/GenBank Feature Table: Definition. Technical Report Version 1.04, DNA Data Bank of Japan, Mishima, Japan; EMBL Data Library, Heidelberg, Federal Republic of Germany; GenBank, Los Alamos, NM and Mountain View, CA, USA.
- Etzold, T. (1992). SRS: Sequence Retrieval System. Technical Report Version 3.0, EMBL, Heidelberg, Germany.
- Hart, K., Searls, D. B., and Overton, G. C. (1993). SORTEZ: A relational translation of NCBI's ASN.1 database. Submitted for publication.
- McAllister, D. and McDermott, D. (1988). Truth maintenance systems. AAAI88 Tutorial Program.
- NCBI Staff (1992). ENTREZ: Sequences Users's Guide. Technical Report Release 1.0, National Center For Biotechnology Information, National Library of Medicine, Bethesda, MD.
- Overton, G. C., Aaronson, J., and Haas, J. (1993). Overview of QGB. Technical report, University of Pennsylvania. In preparation.
- Overton, G. C., Koile, K., and Pastor, J. A. (1989). GeneSys: A knowledge management system for molecular biology. In Bell, G. and Marr, T., editors, *Computers and DNA*, pages 213–240, Reading, MA. Addison-Wesley.

- Pastor, J. A., Koile, K., and Overton, G. C. (1991). Using analogy to predict functional regions on genes. In *Proceedings of the 24th Hawaii International Conference on System Science*, volume I, pages 615–625.
- Searls, D. B. (1988). Representing genetic information with formal grammars. In *Proceedings of the Seventh National Conference on Artificial Intelligence, AAAI-88*, volume 2, pages 386–391, San Mateo, CA. American Association for Artificial Intelligence, Morgan Kauffman Publishers, Inc.
- Searls, D. B. (1989). Investigating the linguistics of DNA with Definite Clause Grammars. In Lusk, E. and Overbeek, R., editors, *Logic Programming: Proceedings of the North American Conference, Vol. 1*, volume 1, pages 189–208. MIT Press.
- Searls, D. B. (1992). The computational linguistics of biological sequences. In Hunter, L., editor, *Artificial Intelligence and Molecular Biology*, chapter 2, pages 47–120. AAAI Press.

List of Tables

I	Query Language Operators. Terminology: <i>feature</i> refers either to the symbolic label for a feature record or a feature record; <i>location specifiers</i> uses essentially the same syntax as that specified in the DEG FEATURE TABLE description manual (DEG Staff, 1992); allowed <i>fields</i> are listed in Table II; <i>pattern</i> is a string or atom; <i>condition</i> is a test which returns true or false.	23
II	Some of the permissible fields and subfields in QGB.	24

List of Figures

1 Simplified flow diagram and functional components of QGB. 25

2 FEATURE TABLE example. The human α -hemoglobin gene (HUMAGL1). 26

3 Idealized view of a parse tree for a typical eukaryotic protein coding gene. 27

4 The parse tree and augmented FEATURE TABLE for the HUMAGL1 α -hemoglobin gene. A) standard representation of the parse tree for HUMAGL1 generated by the SSP; B) a representation of the parse tree for HUMAGL1 as a corrected, augmented FEATURE TABLE. Note that the start position for each interval in the parse tree is one less than the start position in the corresponding FEATURE TABLE entry because the SSP indexes on the space between characters rather than the character itself. 28

Table I: Query Language Operators. Terminology: *feature* refers either to the symbolic label for a feature record or a feature record; *location specifiers* uses essentially the same syntax as that specified in the DEG FEATURE TABLE description manual (DEG Staff, 1992); allowed *fields* are listed in Table II; *pattern* is a string or atom; *condition* is a test which returns true or false.

OPERATOR SYNTAX	DESCRIPTION
Feature/Subsequence Constructors in SELECT and WHERE Arguments:	
JUNC(<i>offset1, feature1, feature2, offset2</i>)	Form a feature record that spans from offset1 to offset2 relative to the junction of feature1 and feature2.
SPAN(<i>offset1, offset2, feature</i>)	Form a feature record that spans the range from offset1 to offset2 relative to the 5'end of <i>feature</i> .
SPAN(<i>feature, offset1, offset2</i>)	Form a feature record that spans the range from offset1 to offset2 relative to the 3'end of <i>feature</i> .
CONCAT(<i>list of features</i>)	Concatenate the ordered set of specified features.
Feature/Subsequence Extraction in SELECT and WHERE Arguments:	
ALL(<i>feature</i>)	Select all sequences labeled feature.
NTH(<i>n, feature</i>)	Select nth sequence labeled feature.
FIRST(<i>feature</i>)	Select first sequence labeled feature.
LAST(<i>feature</i>)	Select last sequence labeled feature.
FROM_TO(<i>m, n, feature</i>)	Select the mth to the nth sequences labeled feature.
Subsequence Extraction in SELECT and WHERE Arguments:	
RANGE(<i>location specifier</i>)	Sequence description in FEATURE TABLE syntax.
Set Operations in SELECT Argument:	
<i>field</i> ∪ <i>field</i>	union of sets.
<i>field</i> ∩ <i>field</i>	intersection of sets.
Comparison in WHERE Argument:	
<i>field</i> {= =\= > < >= =<} <i>test</i>	Compare the <i>field</i> value to the <i>test</i> value. Operators are overloaded on primitive types (integer, real, string as appropriate).
String Matching in WHERE Argument:	
<i>field</i> IN <i>set of values</i>	Succeeds if one of the <i>set of values</i> is equal to an element of <i>field</i> .
<i>field</i> AMONG <i>list of patterns</i>	Succeeds if one of the <i>list of patterns</i> is a substring of an element of <i>field</i> .
Boolean Operations in WHERE Argument:	
<i>condition</i> AND <i>condition</i>	Conjunction.
<i>condition</i> OR <i>condition</i>	Disjunction.
NOT <i>condition</i>	Negation (by failure).

Table II: Some of the permissible fields and subfields in QGB.

FIELD/SUBFIELD	TYPE	DESCRIPTION
entry	<i>record</i>	complete GenBank entry
pt	<i>tree</i>	parse tree generated by SSP
pt:intron	<i>set of records</i>	introns listed and inferred from FEATURE TABLE
pt:exon	<i>set of records</i>	exons listed and inferred from FEATURE TABLE
...	<i>set of records</i>	other features listed and inferred from FEATURE TABLE
ft	<i>set of records</i>	features from GenBank FEATURE TABLE
ft:intron	<i>set of records</i>	introns listed in FEATURE TABLE
ft:exon	<i>set of records</i>	exons listed in FEATURE TABLE
...	<i>set of records</i>	other features listed in FEATURE TABLE
embl	<i>set of records</i>	features from old EMBL features in COMMENTS
embl:intron	<i>set of records</i>	introns listed in old EMBL FEATURE TABLE
embl:exon	<i>set of records</i>	exons listed in old EMBL FEATURE TABLE
...	<i>set of records</i>	other features listed in old EMBL FEATURE TABLE
locus	<i>record</i>	locus field line
locus.id	<i>string</i>	locus ID
locus.len	<i>integer</i>	sequence length
locus.type	<i>string</i>	one of ds-DNA, ss-mRNA or ss-RNA
locus.date	<i>record</i>	last entry update
locus.division	<i>string</i>	flat-file division
definition	<i>string</i>	definition field line
accession	<i>list of string</i>	GenBank accession numbers
keywords	<i>list of string</i>	list of keywords
source	<i>string</i>	description of source
organism	<i>list of string</i>	nodes on path through taxonomy
organism.genus	<i>string</i>	applies to organism of entry
organism.species	<i>string</i>	applies to organism of entry
common_name	<i>string</i>	common name of (selected) organism
articles	<i>record</i>	list of articles as records
article.author	<i>list of records</i>	author names
article.title	<i>string</i>	article titles
article.citation	<i>record</i>	article citation
comments	<i>text</i>	comment field of GenBank entry as text string

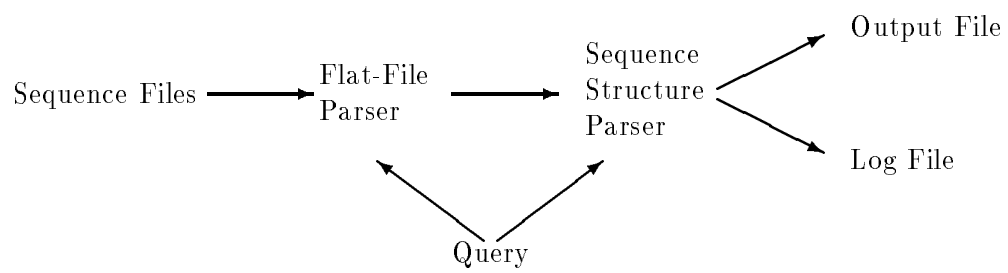


Figure 1: Simplified flow diagram and functional components of QGB.

```

LOCUS      HUMAGL1      1138 bp ds-DNA      PRI      18-JUN-1991
DEFINITION Human alpha-globin germ line gene.
...
FEATURES             Location/Qualifiers
   CDS                join(135..230,348..551,692..820)
                     /codon_start=1
                     /translation="MVLSPADKTNVKAAWGKVG AHAGEYGAEALERMFLSFPTTKTYF
                     PHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLL
                     SHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR"
   precursor_RNA      98..929
                     /note="primary transcript"
   mRNA               98..230
   mRNA               348..551
   exon               692..929
                     /number=3
BASE COUNT      183 a    412 c    350 g    193 t
ORIGIN
      1 aggccgcgcc cgggctccg cgccagccaa tgagcgcgc cggccgggc gtgccccgc
...

```

Figure 2: FEATURE TABLE example. The human α -hemoglobin gene (HUMAGL1).

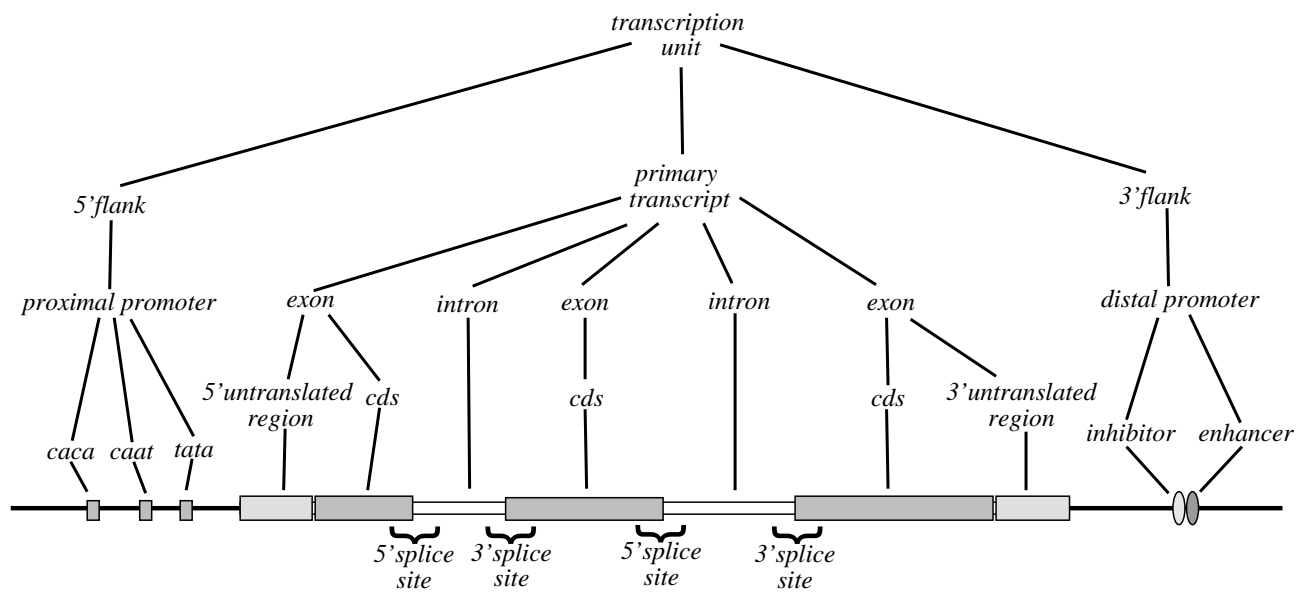


Figure 3: Idealized view of a parse tree for a typical eukaryotic protein coding gene.

<p>A)</p> <pre> cluster(pos(<, [0,0]), pos(>, [1138,1138]), (t_u(pos(<, [0,0]), pos(>, [1138,1138]), (f_f(pos(<, [0,0]), pos(=, [97,97]), (gap(pos(<, [0,0]), pos(=, [97,97]), (p_t(pos(=, [97,97]), pos(=, [929,929]), (exon(pos(=, [97,97]), pos(=, [230,230]), (five_utr(pos(=, [97,97]), pos(=, [134,134]), (gap(pos(=, [97,97]), pos(=, [134,134]), (cds(pos(=, [134,134]), pos(=, [230,230]), (intron(pos(=, [230,230]), pos(=, [347,347]), (gap(pos(=, [230,230]), pos(=, [347,347]), (exon(pos(=, [347,347]), pos(=, [551,551]), (cds(pos(=, [347,347]), pos(=, [551,551]), (intron(pos(=, [551,551]), pos(=, [691,691]), (gap(pos(=, [551,551]), pos(=, [691,691]), (exon(pos(=, [691,691]), pos(=, [929,929]), (cds(pos(=, [691,691]), pos(=, [820,820]), (three_utr(pos(=, [820,820]), pos(=, [929,929]), (gap(pos(=, [820,820]), pos(=, [929,929]), (t_f(pos(=, [929,929]), pos(>, [1138,1138]), (gap(pos(=, [929,929]), pos(>, [1138,1138]), (</pre>	<p>B)</p> <table border="0"> <thead> <tr> <th style="text-align: left;">FEATURES</th> <th style="text-align: left;">Location/Qualifiers</th> </tr> </thead> <tbody> <tr> <td>trans_unit</td> <td><1..>1138 join(5'flank, prim_transcript,3'flank) /gene="alpha-globin"</td> </tr> <tr> <td>5'flank</td> <td><1..97 /label=5'flank</td> </tr> <tr> <td>promoter</td> <td>27..31</td> </tr> <tr> <td>promoter</td> <td>70..72</td> </tr> <tr> <td>prim_transcript</td> <td>98..929 join(exon1,intron1, exon2,intron2,exon3) /label=prim_transcript /note="primary transcript"</td> </tr> <tr> <td>exon</td> <td>98..230 /label=exon1 /number=1</td> </tr> <tr> <td>5'UTR</td> <td>98..134</td> </tr> <tr> <td>CDS</td> <td>join(135..230,348..551,692..820) /label=CDS /codon_start="1 " /translation= NOT SHOWN</td> </tr> <tr> <td>intron</td> <td>231..347 /label=intron1 /number=1</td> </tr> <tr> <td>exon</td> <td>348..551 /label=exon2 /number=2</td> </tr> <tr> <td>intron</td> <td>552..691 /label=intron2 /number=2</td> </tr> <tr> <td>exon</td> <td>692..929 /label=exon3 /number=3</td> </tr> <tr> <td>3'UTR</td> <td>821..929</td> </tr> <tr> <td>pA_sig</td> <td>909..914 /sequence="aataaa" /certainty=0.792</td> </tr> <tr> <td>3'flank</td> <td>930..>1138 /label=3'flank</td> </tr> </tbody> </table>	FEATURES	Location/Qualifiers	trans_unit	<1..>1138 join(5'flank, prim_transcript,3'flank) /gene="alpha-globin"	5'flank	<1..97 /label=5'flank	promoter	27..31	promoter	70..72	prim_transcript	98..929 join(exon1,intron1, exon2,intron2,exon3) /label=prim_transcript /note="primary transcript"	exon	98..230 /label=exon1 /number=1	5'UTR	98..134	CDS	join(135..230,348..551,692..820) /label=CDS /codon_start="1 " /translation= NOT SHOWN	intron	231..347 /label=intron1 /number=1	exon	348..551 /label=exon2 /number=2	intron	552..691 /label=intron2 /number=2	exon	692..929 /label=exon3 /number=3	3'UTR	821..929	pA_sig	909..914 /sequence="aataaa" /certainty=0.792	3'flank	930..>1138 /label=3'flank
FEATURES	Location/Qualifiers																																
trans_unit	<1..>1138 join(5'flank, prim_transcript,3'flank) /gene="alpha-globin"																																
5'flank	<1..97 /label=5'flank																																
promoter	27..31																																
promoter	70..72																																
prim_transcript	98..929 join(exon1,intron1, exon2,intron2,exon3) /label=prim_transcript /note="primary transcript"																																
exon	98..230 /label=exon1 /number=1																																
5'UTR	98..134																																
CDS	join(135..230,348..551,692..820) /label=CDS /codon_start="1 " /translation= NOT SHOWN																																
intron	231..347 /label=intron1 /number=1																																
exon	348..551 /label=exon2 /number=2																																
intron	552..691 /label=intron2 /number=2																																
exon	692..929 /label=exon3 /number=3																																
3'UTR	821..929																																
pA_sig	909..914 /sequence="aataaa" /certainty=0.792																																
3'flank	930..>1138 /label=3'flank																																

Figure 4: The parse tree and augmented FEATURE TABLE for the HUMAGL1 α -hemoglobin gene. A) standard representation of the parse tree for HUMAGL1 generated by the SSP; B) a representation of the parse tree for HUMAGL1 as a corrected, augmented FEATURE TABLE. Note that the start position for each interval in the parse tree is one less than the start position in the corresponding FEATURE TABLE entry because the SSP indexes on the space between characters rather than the character itself.