Object-Oriented Database with Rule-Based Query Interface for Genomic Computation

Susumu Goto¹  Norihiro Sakamoto²  Toshihisa Takagi³

¹Department of Computer Science and Communication Engineering, Kyushu University
Hakozaki, Higashi-ku, Fukuoka 812 Japan
²Kyushu University Medical School
Maidashi, Higashi-ku, Fukuoka 812 Japan
³Human Genome Center, The Institute of Medical Science, The University of Tokyo
Shirokanedai, Minato-ku, Tokyo 108, Japan

ABSTRACT
A large amount of human genome data has been collected and efficient techniques for handling the data and for building and testing biological hypotheses are needed. We developed an object-oriented database system with rule-based query interface for genomic computation. The database is constructed on a commercially available object-oriented database system GemStone and contains GenBank entries as complex objects. To handle complex objects in logical queries, concepts of meta variables and meta atoms are adopted in rule-based query interface. This system is powerful to construct database queries from simple keyword search to various biological hypothesis.

1 INTRODUCTION
The Human Genome Project is directed towards experimentally determining all the sequences of the human genome and acquiring knowledge on the structure, function and evolution of the human genome. The genome of an organism is a set of chromosomes that contains all of its genetic information. The human genome comprises twenty four chromosomes that consists of about three billion base pairs.

There are various information concerning the genome, which we call the genomic information. For example, DNA sequences consisting of four kinds of bases (adenine, thymine, guanine and cytosine — symbolically represented as A, T, G and C respectively), protein sequences consisting of twenty kinds of amino acids, 3-D structures of proteins and mapping data of a gene to a certain position of the chromosomes are the genomic information. Figure 1 shows relationships between some of the genomic information.

To achieve the goals of the project, a large amount of the genomic information is rapidly increasing. There are several data banks to maintain the genomic information, for example GenBank¹ for DNA sequences, PIR² for protein sequences, PDB³ for 3-D structures of proteins. Some of them support the on-line services for searching based on keywords and homology search (similarly search between sequences such as FASTA⁴). But many complex searches such as the combination of the above ones, the users have to write programs for their own use.

The project requires not only the accumulation of the genomic information but also the analyses of them. Analysis consists of constructing and testing a hypothesis and additional analyses of the results. We call the iteration of the analyses the genomic computation. For example, the following are genomic computation.

- Searching biologically significant sequences from the data banks.
- Predicting the function or the structure of the experimentally obtained sequences (e.g., by similarity search to the known sequences).
- Getting together the information of a gene usually stored separately in several entries. (An example of GenBank entry is shown in Appendix.)
- Getting together the related information over several data banks.

Most of genomic computation requires the user's program such as SIGNAL SCAN⁵. Therefore, database systems supporting interfaces to easily construct and test the hypotheses has to be designed.

In 1992, we developed a relational database system ODS that contains GenBank data⁶. We also integrated deductive functions to it. The system is useful for the signal sequence search or predicting structures of sequences. (Signal sequences are the one of the important ones related to regulatory proteins.) However, the system does not store all of the information in GenBank due to its various and complex structures.

65
GenBank contains keywords for retrieving entries, references in which the sequence is reported, biological features of the sequence, etc., in addition to the sequence data. (An example of GenBank entry is given in Appendix.) The other data banks also contain various kinds of information. The information in the data banks is complex (e.g., biological features and references in GenBank). The object-oriented database (OODB) system is suitable to manage the complex information because it can manage various types of data such as nested relations and long text data like DNA sequences.

We developed an OODB system managing all of the information in GenBank. The system raveled out the inefficiency of ODS, which came from the frequent join operations. We integrated the deductive function for easily constructing and testing hypotheses to the OODB system. Recently, there has been much research in developing logical languages into OODB systems (for example, HiLog®, LLO®). One main topic of the research seems to be the formal discussion of the languages. Our approach is to integrate deductive functions from the practical point of view.

We describe below our OODB system with rule-based query interface for genomic computation. The system contains GenBank data. We show that the OODB systems and their additional deductive function are useful for the genomic computation.

The rest of this paper is organized as follows. Section 2 describes features of genomic information and some classes for them in our OODB system. Section 3 introduces the rule-based query interface of our system. The syntax of the query language is described there. In section 4, we describe the overview of the system. Section 5 concludes this paper.

2 GENOMIC INFORMATION IN OBJECT-ORIENTED DATABASE

Human genome data has been reported (17,752 entries; 18,876,706 bases in GenBank release 73. September. 1992). These data can be obtained in a text file format. GenBank has been internally managed by a relational database (RDB) system. However the services of GenBank are limited to either retrieving sequences based on keywords or homology search. Retrievals of sequences based on biological features requires a new program.

In this section, we describe difficulties in managing the genome data by RDB systems and our approach to managing them by our OODB system.

2.1 Difficulties in Managing Genomic Information by RDB Systems

We developed the system ODS with a high level query processing function. The system integrates information on biological features into the search for signal sequences by using a RDB system and processes rule-based queries by applying a deductive engine to it.

Figure 2 shows an example of genomic information stored in the ODS system. The human DNA sequences kept in GenBank vary from less than 10 base pairs to more than 7 kilobase pairs in length. This variety in length is one of the drawbacks to store and search for signal sequences in a database. The ODS copes with this problem by dividing sequences into the 8 bases long overlapping oligonucleotides (Figure 2: Table Sequences).

It is difficult to store all information of FEATURES TABLE that describes biological features of genes because of complexity of its structure. The ODS stores the biological features in the Table Features (Figure 2: Table Features) and the table is linked to the other tables by the key attribute Locus. Some biological features could not be stored in the table for reasons of complexity. In the research on the genomic information, DNA sequences and their biological features are frequently needed, which results in the frequent join operations of these tables.

There are some interrelated entries, because the relationship between GenBank entries and genes is not one to one. There is the case where several entries belong to one gene. Some DNA entries also have mRNA entries and protein data that originate from them. Hence, the database
Table Keyword

<table>
<thead>
<tr>
<th>Locus</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGMPPINS</td>
<td>preproinsulin</td>
</tr>
<tr>
<td>AGMPPINS</td>
<td>preproinsulin</td>
</tr>
</tbody>
</table>

Table Features

<table>
<thead>
<tr>
<th>Locus</th>
<th>Start</th>
<th>End</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGMPPINS</td>
<td>426</td>
<td>463</td>
<td>5'UTR</td>
</tr>
<tr>
<td>AGMPPINS</td>
<td>654</td>
<td>670</td>
<td>5'UTR</td>
</tr>
<tr>
<td>AGMPPINS</td>
<td>426</td>
<td>463</td>
<td>exon</td>
</tr>
<tr>
<td>AGMPPINS</td>
<td>654</td>
<td>857</td>
<td>sig_peptide</td>
</tr>
<tr>
<td>AGMPPINS</td>
<td>671</td>
<td>742</td>
<td>exon</td>
</tr>
</tbody>
</table>

Table Sequences

<table>
<thead>
<tr>
<th>Locus</th>
<th>Start</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGMPPINS</td>
<td>1</td>
<td>GGGCCATC</td>
</tr>
<tr>
<td>AGMPPINS</td>
<td>2</td>
<td>GGGCCATCC</td>
</tr>
</tbody>
</table>

Figure 2: Examples of genome data in relational tables (corresponding to the entry in Appendix)

The OODB system for the genomic computation has to provide functions of handling data structure that can store an attribute for the set of entries. And the entries have to be not only the ones in GenBank but also the ones in the other data banks.

2.2 Management of Genomic Information by an OODB System

The OODB consists of objects and linkages between them. An object consists of attributes and methods. Figure 3 shows the classes (represented by rectangles) in the database of genomic information and their linkages. Linkages between attributes and class are represented by arrows and class hierarchy is represented by solid lines. Attribute names are represented in italics.

The references between sequences and their biological features can be easily achieved, because the references from an object to other related objects using pointers are easy in OODB systems. Built-in methods (e.g., for the homology search) reduce the user's burden of programming for a combination of keyword search and homology search.

In the next subsection, we describe data structure for genomic information in detail.

2.3 Classes of Genomic Information in the OODB System

The most important class for GenBank is the class GenBankEntry, which corresponds to the GenBank entry and contains all data fields (LOCUS, DEFINITION, etc., in the example in Appendix) as its attributes. The attributes id and sequence are necessary for the data not only in GenBank but also in the other data banks. Those attributes are inherited from the class DNAEntry. Classes for the other data banks could be easily designed as the subclasses of the class DNAEntry.

There are attributes defined as complex objects such as references and featureTable (Figure 3: (a)). The attribute featureTable in the class GenBankEntry is defined as a set of instances of the class FeatureTuple. The class FeatureTuple is the one for biological features of the sequences. As described in the previous section, they are important for the researchers and are very complex. It has locationDescriptor, feature and qualifiers as attributes. The value of the attribute feature is an instance of the class Feature. The class Feature has featureKey for 'exon,' '5'UTR,' etc., and the set of pointers to the GenBankEntry in which the instance of featureKey is included. The attribute featureQualifiers is a set of instances of the class Qualifier that describes complex objects.

The value of the attribute locationDescriptor is the instance of the class LocationDescriptor. This is either the instance of the class SimpleLocation or the instance of the class ComplexLocations such as 'join(426..463,654..670),' as shown in Appendix. The class Location contains the attributes startPos and endPos representing the regions of biological features in the nucleotide sequence (Figure 3: (b)). These classes have the methods length and toSequence. The method length returns the length of the location and the method toSequence returns the corresponding
In the OODB systems, the users have to program with the procedural language like C++ to describe a complex query. In research on the genomic information, it is necessary to easily construct and test hypotheses and represent a higher structure of DNA or protein. The declarative expression of rule-based queries minimizes the programming effort for constructing hypotheses.

To integrate deductive functions in the OODB systems, we meet the following problems.

- How can complex objects in the forms of facts be managed?
- How can methods be implemented as the deductive functions?

To cope with these problems, logical languages for OODB systems have been proposed (for example, HiLog', LL08). Our approach is the one from the practical point of view rather than formal languages discussed in the previous research.

We propose a rule-based query interface for the OODB system. The rules contain the meta variable for a set of objects and can define the methods applied to each instance in the set. Although it is a subset of those languages, it is powerful for the genomic computation. In this section, we introduce the overview of the interface. The proposed query processing function can handle rules in the current fixed class hierarchy, because the database for the genomic information merely requires update for data structure. The update of data is not also considered.

### 3.1 Examples of Rules in the Interface

We introduce the notion of a meta variable and a meta atom. A meta variable represents an object or a set of objects. A meta atom includes a meta variable in its predicate symbol and plays the role of message passing to each object in the set represented by the meta variable. The syntax of the rules is extended from the first order form in the following constructs.

**Meta Variables:** Meta variables are terms. Meta variables appear in the meta atoms (described later). A meta variable represents a set of objects and plays a role of extracting elements of the set.

**Class Atoms:** If $A$ is a class name in which there are $n$ instance variables $i_1, \ldots, i_n$, then $A(i_1, \ldots, i_n)$ is an atom. We call this a class atom. We consider the inheritance of instance variables and method defined in the class. We discuss about them in the end of this section.

**Meta Atoms** Suppose $M$ is a meta variable that represents a set of objects in the class $C$, and $D$ is a predicate derived from a class redefinition rule or $C$ itself.
A class redefinition rule is the rule for extracting only required instance variables from a certain class. We describe it later. Then $M/D(i_1, \ldots, i_k)$ is an atom. We call this a meta atom. The intuitive meaning of the meta atom is the message passing to each object in $M$ to obtain the values of instance variables represented by its arguments.

If a rule has only one body literal and the literal is a class atom, then the rule is called a class redefinition rule. This rule redefines the class represented by $L_i$ to hold only the necessary instance variables.

Example 1: Consider the classes Feature and GenBankEntry as mentioned in the previous section. First, we retrieve the objects in the class Features where featureKey is 'CAAT-signal.' Second, we retrieve instances in the class GenBankEntry that contain the objects retrieved from the class Features. Finally, we obtain locusNo, locusName and features of the retrieved objects, and as for features, we obtain location and featureKey.

Although the GenBank on-line service supports entry search based on keywords or locus name, it does not support the one based on the biological features. This search is one of the important tasks in the research on the genomic information (for example, signal sequence search mentioned in Section 1). This example, which is represented by the following rules, shows one of the useful points of the system.

\[
\begin{align*}
\text{feature1}(Entry, Key):&= \quad \text{Feature}(Entry, Location, Key, Qualifier). \quad (1) \\
\text{feature2}(Location, Key):&= \quad \text{Feature}(Entry, Location, Key, Qualifier). \quad (2) \\
\text{entry}(No, Name, Features):&= \quad \text{GenBankEntry}(No, Name, AccNo, Def, Keywords, Ref, Features, Seq). \quad (3) \\
\text{query}(LocusNo, LocusName, Location, Key):&= \quad \text{feature1}(Entries,'CAAT-signal'), \\
&\quad \text{Entries}/entry(LocusNo, LocusName, Features), \\
&\quad \text{Features}/feature2(Location, Key). \quad (4)
\end{align*}
\]

The rules (1)~(3) are the class redefinition rules. Entries and Features in the rule (4) are meta variables, and they represent a set of instances of the class GenBankEntry and Feature, respectively. The literal \text{feature1}(Entries,'CAAT-signal') corresponds to the selection operation to the database. The literals \text{Entries}/entry(LocusNo, LocusName, Features) and \text{Features}/feature2(Location, Key) correspond to the message passing to the instances of the class GenBankEntry and to the instances of the class Feature, respectively.

Example 2: In GenBank, there is the case where several entries represent one gene. Thus it is important to investigate the correspondence between entries and genes. This example shows one of such tasks constructing the hypothesis that the entries represent same genes under the following condition.

1. The entries have same keywords. In this example the keyword is 'cystic fibrosis.'
2. The sequences of the entries have similarity over the 95% matches by the homology search such as FASTA.
3. If entry A and entry B, and entry B and entry C represent same gene, respectively, then entry A and entry C also represent same gene.

\[
\begin{align*}
\text{cf-features}(Locus, Features):&= \quad \text{GenBankEntry}(Locus, Acc, Def,'cystic fibrosis', Ref, Features, Sequence). \quad (5) \\
\text{cf-sequence}(Locus, Sequence):&= \quad \text{GenBankEntry}(Locus, Acc, Def,'cystic fibrosis', Ref, Features, Sequence). \quad (6) \\
\text{cf-exon}(Locus, Seq):&= \quad \text{cf-features}(Locus, Features), \\
&\quad \text{Features}/feature(Entries, Region,'exon'), \\
&\quad \text{Region}/Location(Start, End, Length, Seq). \quad (7) \\
\text{same-gene}(Locus1, Locus2):&= \quad \text{cf-sequence}(Locus1, Sequence1), \\
&\quad \text{cf-sequence}(Locus2, Sequence2), \\
&\quad \text{homology}(Sequence1, Sequence2, Similarity), \\
&\quad \text{Similarity} > 95.0. \quad (8) \\
\text{same-gene}(Locus1, Locus2):&= \quad \text{cf-exon}(Locus1, Sequence1), \\
&\quad \text{cf-exon}(Locus2, Sequence2), \\
&\quad \text{homology}(Sequence1, Sequence2, Similarity), \\
&\quad \text{Similarity} > 95.0. \quad (9) \\
\text{same-gene}(Locus1, Locus2):&= \quad \text{cf-sequence}(Locus1, Sequence1), \\
&\quad \text{cf-sequence}(Locus2, Sequence2), \\
&\quad \text{homology}(Sequence1, Sequence2, Similarity), \\
&\quad \text{Similarity} > 95.0. \quad (10) \\
\text{same-gene}(Locus1, Locus2):&= \quad \text{same-gene}(Locus1, Locus), \\
&\quad \text{same-gene}(Locus, Locus2). \quad (11) \\
\text{same-gene}(Locus1, Locus2):&= \quad \text{same-gene}(Locus1, Locus), \\
&\quad \text{same-gene}(Locus, Locus2). \quad (12)
\end{align*}
\]

The rules (5) and (6) retrieve entries that have the keyword 'cystic fibrosis.' By using rule (5), the locus names and features of the retrieved entries are obtained. Similarly, sequences of the retrieved entries are obtained using the rule (6). The rule for \text{cf-exon}(LocusName, Sequence) retrieves the exon subsequence of the entry (designated by LocusName). \text{same-gene}(LocusName1, LocusName2) means the entry whose LocusName is LocusName1 and the entry whose LocusName is LocusName2 can be the same gene, because similarity of the two sequences exceeds 95%. The predicate \text{homology} is built-in. This homology search function can be implemented as a built-in method in the OODB system using the FASTA program.

In the above examples, the class atom has only instance variables in its arguments. The implementation of the class atom uses the message passing to the instance of the class even though the arguments represent only the
instance variables. Thus some methods can be also integrated in the arguments. For example, we can extend the class atom NucleotideSeq (Sequences) to the atom NucleotideSeq(Sequences, Length, Composition). The meanings of the arguments Length and Composition are the message passing to the instance of class NucleotideSeq corresponding to the methods length and composition respectively.

4 SYSTEM DESCRIPTION

Figure 4 shows an overview of our system. We use the commercially available object-oriented database system GemStone (version 3.0) and its Smalltalk Interface. We use the primate data file gbpri.seq (about 50MB) in GenBank release 73. The program that transforms the GenBank file into the data for GemStone is written in ObjectWorks\Smalltalk R4.1.

Users can give either the primitive queries or the rule-based queries. For the primitive queries such as searching based on keywords or/and biological features, the system supports a window interface. For the rule-based queries such as the ones described in the previous section, the rule transformer transforms them into the Smalltalk queries. Both interfaces are written by ObjectWorks\Smalltalk R4.1.

Size of the database (data file and log file) is about 130MB that is about 2.6 times larger than original flat file. It takes about 15 second to retrieve objects in the class GenBankEntry that contains the feature key “CAATsignal”, and then their locus name, feature key, and the location of the key, which is like the one in the example in section 3.

5 CONCLUSION

The database for the genomic information is required to handle various kinds of information obtained by biological experiments. They are not easy to deal with for the relational or deductive database systems. Therefore, we developed the object-oriented database system for the genomic information by using its ability to handle various data types. We integrated the deductive function to the system for the genomic computation. This system is powerful to construct database queries from simple keyword search to various hypotheses.

One of our future work is the integration of the other data banks. For example, it is necessary to classify GenBank entry by using mapping information because there are many cases that GenBank data is not sufficient for clarifying the relationship between entries and genes. For instance, the integration of the information in GDB (mapping information of a gene or locus to a certain position of the chromosomes) is important.

We consider that the OODB system is suitable for this integration because of the following reasons:

- The design and implementation of the new class are relatively easy by using existing class (for example the class DNAEntry or the class Seq)
- The known relationship between the GenBank entry and the information in GDB is easily implemented by linking them using pointers.

We also consider that the deductive function, which enables users to easily construct hypotheses, is powerful to describe the unknown relationship between the information and to test the hypothetical relationship.

We use data from the GenBank flat file. The distribution of GenBank data in ASN.1\footnote{1) Burks,C., Cassidy,M., Cinkosky,M.J., Cumella,K.E., Gilna,l Hayden,J.E.-D., Keen,G.M., Kelley,T.A., Kelly,M., Kristoffersen,D. and Ryals,J., “Genbank,” Nucleic Acids Research, 19, pp 2221-2225(1991) format (Figure 5) will soon be available and we will use these data.

ACKNOWLEDGEMENTS

We thank Mariko Ohara for reading the manuscript. This work has been supported in part by Grant-in-Aid (04261101) for Scientific Research on Priority Areas from The Ministry of Education, Science and Culture.

REFERENCES

Figure 5: An example of genome data in ASN.1 format


8) Lou,Y. and Ozsoyoglu,Z.M., “LLO: An Object Oriented Deductive Language with Methods and Methods In-
## Appendix: An example of GenBank entry

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCUS</td>
<td>AGMPPINS</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>C. aethiops gene for preproinsulin</td>
</tr>
<tr>
<td>ACCESSION</td>
<td>X61092</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>insulin; preproinsulin.</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Cercopithecus aethiops DNA.</td>
</tr>
<tr>
<td>ORGANISM</td>
<td>Eukaryota; Animalia; Chordata; Vertebrata; Mammalia; Theria; Eutheria; Primates; Haplorhini; Catarrhini; Cercopithecidae; Cercopithecinae.</td>
</tr>
<tr>
<td>REFERENCE 1</td>
<td>Bell, G. Unpublished (1991)</td>
</tr>
<tr>
<td>STANDARD 1</td>
<td>Full automatic</td>
</tr>
<tr>
<td>REFERENCE 2</td>
<td>Seino, S., Bell, G.I. and Li, W.</td>
</tr>
<tr>
<td>TITLE 2</td>
<td>Sequences of primate insulin genes support the hypothesis of a slower rate of molecular evolution in human and apes than in monkeys</td>
</tr>
<tr>
<td>JOURNAL 2</td>
<td>Unpublished (1991)</td>
</tr>
<tr>
<td>STANDARD 2</td>
<td>Full automatic</td>
</tr>
<tr>
<td>REFERENCE 3</td>
<td>Seino, S., Bell, G.I. and Li, W.-H.</td>
</tr>
<tr>
<td>TITLE 3</td>
<td>Sequences of primate insulin genes support the hypothesis of a slower rate of molecular evolution in humans and apes than in monkeys</td>
</tr>
<tr>
<td>STANDARD 3</td>
<td>Full staff_review</td>
</tr>
<tr>
<td>COMMENT</td>
<td>*source: is_macronuclear=N; *source: is_proviral=N; *source: is_germline=N; From EMBL entry CEPPINS; dated 12-Nov-1991.</td>
</tr>
</tbody>
</table>

### FEATURES

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5'UTR</td>
<td>Location/Qualifiers</td>
</tr>
<tr>
<td>mat_peptide</td>
<td>join(426..463, 654..670)</td>
</tr>
<tr>
<td>mRNAs</td>
<td>join(426..463, 654..857, 1607..1825)</td>
</tr>
<tr>
<td>CDS</td>
<td>join(671..857, 1607..1752)</td>
</tr>
<tr>
<td>sig_peptide</td>
<td>/product=&quot;Preproinsulin&quot;</td>
</tr>
<tr>
<td>introns</td>
<td>base 1 to 1909</td>
</tr>
<tr>
<td>exons</td>
<td>base 1 to 1909</td>
</tr>
<tr>
<td>3'UTR</td>
<td>base 1 to 1909</td>
</tr>
</tbody>
</table>

### BASE COUNT

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN</td>
<td>338 a 612 c 606 g 353 t</td>
</tr>
<tr>
<td>1</td>
<td>gggcatcca tggggcatttc tggggcatact tggggcaaccc agggactgag aacaggggcc caggacagg</td>
</tr>
<tr>
<td>61</td>
<td>ggtctggggctgagctgtgggctgag ctgggcaaccc ccctgcacgg ccagctctcc tggtgttaatag</td>
</tr>
<tr>
<td>121</td>
<td>tgggaatgg gccggggagg gcttggtctct gctggagaca ttggcggcggc agctgcgagcc</td>
</tr>
<tr>
<td>1741</td>
<td>tctggagact gatctccgagc tgggaggagc gcccacccct cccacctcgc cccacaggg</td>
</tr>
<tr>
<td>1801</td>
<td>gatctgaata gacccctgaa ctaacccgtgc tgggcatctct tttttgggcc cttgcagggc</td>
</tr>
<tr>
<td>1001</td>
<td>cttgcgggag cactgtggta agccctcagc agctctcttc agcttccttc</td>
</tr>
</tbody>
</table>

//