Integration of Genome Databases Using a Deductive Object-Oriented Database

Susumu Goto¹ Norihiro Sakamoto² Toshihisa Takagi¹
goto@ims.u-tokyo.ac.jp nori@csce.kyushu-u.ac.jp takagi@ims.u-tokyo.ac.jp

¹Human Genome Center, Institute of Medical Science, University of Tokyo 4-6-1 Shirokanedai, Minato-ku, Tokyo 108 Japan
²Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812 Japan

Abstract

Recently, many genome databanks were developed as a result of growing genome project activities. Each of them consists of a large amount and variety of data, and they were developed independently. Therefore, their integration and efficient management of the data are required. It is also necessary to develop a framework for easily building and testing biological hypotheses with the integrated database. We developed a deductive object-oriented database for searching an integrated database, acquiring new knowledge from it, and storing the knowledge in the database. It consists of an object-oriented database that integrates the conventional genome databases such as GenBank, and deductive language interface for genome analysis. In this paper, we present an overview of the system and examples of analyses using the database.

1 Introduction

Recently many databanks were developed for genome researchers. In [1], Kamel reported 102 genome databanks related to molecular biology research. The following are examples of them.

- databanks for nucleotide sequences of DNA and RNA (GenBank[2], EMBL[3], DDBJ[4])
- databanks for amino acid sequences of proteins
  - primary sequences (PIR[5], SwissProt[6])
  - motif: biologically significant amino acid sequence patterns (PROSITE[7])

¹五斗進, 高木利久: 東京大学医科学研究所ヒトゲノム解析センター, 〒108 東京都港区白金台 4-6-1
²坂本憲広: 九州大学工学部情報工学科, 〒812 福岡市東区箱崎 6-10-1
- a databank for atomic coordinate and crystallographic structure data of proteins (PDB[8])
- a databank for physical and genetic maps of human genes (GDB[9])
- a databank for bibliographic information (MEDLINE)

Each of these databanks consists of a huge volume and variety of data. For instance, GenBank contains about 400 mega bytes in a flat file format, and includes biological features of sequences, keywords and bibliographic data in addition to nucleotide sequence data.

The databanks are associated with three problems. First, most of them are stored in relational databases or flat files. Because the data are often complicated and nested as shown in Fig.1, it is difficult for users to retrieve detailed information in entries.

Second, they were developed and managed independently. Although some databanks have cross references, the databank users must write programs to search over several databanks by using the cross references.

The third problem is that there is no framework for flexible analyses of genome data, such as building and testing biological hypotheses. The conventional genome databanks restrict the users to only two kinds of searches:

- homology search: search for similar sequence data to a given sequence.
- keyword search: search for entries that include given keywords. An entry of a database corresponds to one unit of experimentally obtained data, which does not necessarily correspond to a gene.

For further analyses of their results, the users must search another database or write programs. Therefore, a framework for easily analyzing genome data in databases is required.

Recent database systems developed for the genome data emphasize the use of object-oriented databases (OODBs) for managing the data and integrating the databanks. The Entrez system is one of those systems and was developed to integrate GenBank, PIR and MEDLINE. The data in the system are described in ASN.1 that is a data description language with object-oriented concept.

The systems developed using OODB are powerful to manage complex data and to integrate several databanks. However, they do not provide a flexible query expression to perform the above analyses. The users can not even define the relationships between existing data.

For example, the following typical flow of genome analyses shows the necessity of the flexible query expression. Assume that we determined sequences experimentally (containing 300~500 nucleotides, respectively). First of all, we must assemble the sequences to construct a contiguous sequence that represents a whole gene (usually containing several thousands of nucleotides). Then we will search a nucleotide sequence databank for homologous sequences to know what kind of biological function it possesses. In case that we are interested in proteins, we must search an amino acid sequence databank by translating the nucleotide sequence into the amino acid sequence. When we want to know the regulation of the gene (the sequence), we must check whether the retrieved homologous sequence has biological features related to the regulation, and must predict the secondary structure of the original sequence to check whether it has a secondary structure related to the regulation. Although this flow of analyses is a typical and routine work, it consists of many kinds of programs or algorithms, and therefore very complex. Hence it is much more complicated to repeat the analysis with changing parameters or algorithms and perform further analyses, for instance, to deduce knowledge on evolution.
Figure 1: A part of a GenBank entry
Biological features (FEATURES) and a sequence (ORIGIN) extracted from the entry of HSA1MBG1 locus. The FEATURES field contains variety of data related to biological features of the sequence, such as keys (exon, intron, ...), their locations, produced proteins, translated amino acid sequences, etc. Joined regions for mRNA and CDS make the data structure complicated. Some features refer to other data entries. In this case, CDS refers to the entries with accession numbers X54817 and X54818. It makes the data further complicated.
Flexible analyses should be easily performed in genome databases. A deductive language can provide interface to perform such analyses. We developed a deductive object-oriented database for searching the integrated database, defining new relationships between data, and storing them. It consists of the object-oriented database and a deductive language interface.

2 Method and System

2.1 Deductive Object-Oriented Databases

A deductive object-oriented database (DOOD) is an integration of a deductive database (DDB) and an object-oriented database (OODB).

Deductive Databases DDBs are extensions of relational databases (RDBs). They are based on the first order predicate logic and provide users with declarative query interface including recursive queries. In addition to the declarativeness, the data, queries and programs (rules) have a unique Prolog-like form, therefore it is easy for users to define relationships between data. DDBs can handle nested relations by using function symbols. However it is not sufficient to model real world data such as genome data.

Object-Oriented Databases OODBs were developed for handling variety of data types. They can naturally handle complex objects such as FEATURES in GenBank. However, they do not provide declarative query languages. Users must program in a procedural language and it is not easy to define a relationship between the data in an OODB.

Deductive Object-Oriented Databases DOODs are studied for providing both advantages of DDBs and OODBs, i.e. for providing a declarativeness, variety of data types and a logical foundation. There have been many studies on DOODs[10, 11]. Their approaches are mainly theoretical ones. They are intended to integrate object-oriented paradigms, such as classes (for complex objects), inheritances, methods and encapsulation, into DDBs. However there are few systems and applications.

Our Approach We designed and implemented classes for GenBank data in an object-oriented database[12]. Fig.2 shows a part of the classes (We are now integrating GDB and PIR data). Our approach is to develop a declarative query interface to the database for the easy definition of the relationships between entries, such as homologous sequences between a nucleotide sequence databank and an amino acid sequence databank (User Database in Fig.2).

The following are necessary to provide such an interface. (1) The query language should be implemented on an OODB because of the reason we mentioned in section 2. (2) It should be accessible to all objects in the OODB, not only instances of classes but also classes and methods. (3) It should be declarative in the sense that we can easily define and execute queries, and redefine a new kind of queries.

We developed a deductive object-oriented language to meet the above requirements. The language is the extension of the first order logic programming language. The language is different from the first order logic programming language in that concepts of class atoms and method terms are introduced into the language.

A class atom is used to obtain a set of objects that are the instances of the class specified the atom, and defined as follows:

\[ \text{ClassName}(X):=\neg X \text{ is an instance of the class } \text{ClassName}. \]
It corresponds to a message passing to the class.

A method term is used for sending a message to an object. We use the form `object.method`, where `object` is either an object or a variable to be substituted by an object, and `method` is a method defined in the class whose member the object is or a method inherited from its superclasses. It is usually used to obtain an attribute value of the object.

We also implemented a built-in predicate for homology search. Using the predicate, we can easily construct a complex query, for example by combining homology search and keyword search.

### 2.2 System Overview

Fig. 3 shows an overview of our system. We designed and implemented classes for the primate data in GenBank release 78 by using the GemStone OODBMS (version 3.1) and its Smalltalk interface. A part of the classes is shown in Fig. 2.

The users of the system write queries in the deductive language or search an object using a
For the primitive queries such as searching based on keywords, biological features and their combinations, we developed a graphical interface. The queries written in the deductive language are transformed into the Smalltalk queries. Both interfaces are written in ObjectWorks/Smalltalk R4.1.

### 3 An Example

Assume that we have an experimentally obtained nucleotide sequence and we want to know its function. Fig.4 shows the analyses required to determine the function of the sequence. In this case, we search over two databases, GenBank and PIR. Below, we show the required analyses following the arrows numbered (1) ~ (10) in Fig.4.

1. Homology search to GenBank: There are some tools for homology search such as FASTA\[13\] and BLAST\[14\]. We can define the search as a built-in predicate. Here, we use

   \[
   \text{homology(Sequence, Database, Result)}
   \]

   where \text{Sequence} is a given sequence, \text{Database} is a database on which we will perform homology search (in this case GenBank, and in the case of (5) and (6), PIR), and \text{Result} represents an object that has attributes such as entry, score, etc. The following rule to search an entry whose sequence has a homology to the given sequence can be defined.

   \[
   \text{genbankHomology(Sequence, Entry)}:--
   \text{homology(Sequence, genbank, Result)},
   \text{Entry} = \text{Result.entry}.
   \]
(2),(3) Each entry in GenBank has attributes keyword, feature, sequence, etc. Thus, you can get keywords from the entries obtained in (1) by using the following rule.

\[
\text{keywordHomology(Sequence, Keyword):=} \\
\text{genbankHomology(Sequence, Entry),} \\
\text{Keyword = Entry.keyword.}
\]

We can also obtain the biological features of the sequence in the entry in the same way.

(4) We can select sequences that have protein coding regions from the entries obtained in (1).

\[
cds(Sequence, CDS):= \\
\text{genbankHomology(Sequence, Entry),} \\
\text{Feature = Entry.feature,} \\
\text{Feature.key = "CDS",} \\
\text{CDS = Feature.sequence("CDS").}
\]

where sequence is a method defined in the class Feature, which returns a subsequence of the sequence in the entry corresponding to the feature key specified with its argument.

(5) Homology search to PIR: BLAST provides automatic translation of the given nucleotide sequence to the corresponding amino acid sequences and homology search to the amino acid sequence database. Therefore we can define homology search to PIR in the same way as (1).

\[
pirHomology(Sequence, Entry):= \\
\text{homology(Sequence, pir, Result),} \\
\text{Entry = Result.entry.}
\]
(6) Using the results of (4), we can define amino acid sequences that are similar to the sequences translated from the homologous nucleotide sequences.

\[
\text{pirHomology} (\text{Sequence}, \text{Entry}) : - \\
\text{cds}(\text{Sequence}, \text{CDS}), \\
\text{homology}(\text{CDS}, \text{pir}, \text{Result}), \\
\text{Entry} = \text{Result}.\text{entry}.
\]

(7) we can obtain keywords from PIR entries in the same way as (2).

\[
\text{keywordPIRHomology} (\text{Sequence}, \text{Keyword}) : - \\
\text{pirHomology}(\text{Sequence}, \text{Entry}), \\
\text{Keyword} = \text{Entry}.\text{keyword}.
\]

(8)~(10) Using the above results, we can define the function of the sequence in various way. For instance, we can define it as the conjunction of GenBank and PIR keywords.

\[
\text{derived_functions} (\text{Sequence}, \text{Function}, \text{Feature}) : - \\
\text{keywordHomology}(\text{Sequence}, \text{Function}), \\
\text{keywordPIRHomology}(\text{Sequence}, \text{Function}).
\]

It is easy to redefine homology search to GenBank. Using recursive rule, we can obtain more entries than (1). In addition to the rule in (1), we can define the following rule.

\[
\text{genbankHomology}(\text{Sequence}, \text{Entry}) : - \\
\text{cds}(\text{Sequence}, \text{CDS}), \\
\text{genbankHomology}(\text{CDS}, \text{Entry}).
\]

We can also easily define similar functions as a hypothesis using the results of (2) and (3):

\[
\text{similar_function}(\text{Func1}, \text{Func2}) : - \\
\text{keywordHomology}(\text{Sequence}, \text{Func1}), \\
\text{keywordHomology}(\text{Sequence}, \text{Func2}).
\]

Of course, we can use the results of (5), (6) and their combinations as the body literals.

4 Conclusions

We developed a deductive object-oriented database for genome analysis. It consists of an object-oriented database for genome data and a deductive language for a flexible query interface.

If the users want to perform analyses like the examples in section 3 using genome databases that are managed by flat files or other conventional techniques, they must enter commands for FASTA or BLAST, and to obtain a database entry, then edit the resulting file for later analyses. Otherwise, the users must write programs.

In the object-oriented database, users can define various methods for these commands such as homology search and obtaining the sequence in the coding region. However, they also must write programs. In our system, the Smalltalk code for the genbankHomology recursive rules in Example 1 would be the same as that shown in Fig.5, which is more complicated than the deductive rules.
genbankHomologyInit: aSequence
  | cds results |
  entries := Set new.
  cds := Set new.
  results := self blast: aSequence db: 'genbank'.
  results do: [ :result |
    | entry |
    entry := result entry.
    entries add: entry.
    (entry features) do: [ :feature |
      (feature key) = 'CDS' ifTrue: [cds add: (feature sequence: 'CDS')]].
  ].
  cds isEmpty
  ifFalse: [self genbankHomology: cds].

"entries

genbankHomology: aSeqSet
aSeqSet do: [ :aSeq |
  | tmpEntries cds |
  tmpEntries := Set new.
  cds := Set new.
  results := self blast: aSeq db: 'genbank'.
  results do: [ :result |
    | entry |
    (entries includes: entry)
    ifTrue: [tmpEntry add: entry
      entries add: entry]].
  tmpEntry isEmpty ifFalse: [
    tmpEntry do: [ :entry |
      (entry features) do: [ :feature |
        (feature key) = 'CDS'
        ifTrue: [cds add: (feature sequence: 'CDS')]].
      cds isEmpty ifFalse: [self genbankHomology: cds]].

"entries

Figure 5: An example of the Smalltalk code for the recursive rules in Example 1.

Our object-oriented language does not support all object-oriented features like LLO[11], such as method and class definition. The features all depend on the object-oriented database framework, and our language can call a method from an object-oriented database and search for objects. It provides a simple and powerful way of searching for objects and building biological hypotheses.

Various works are still required in the future. It is necessary to develop a framework for storing rules and their results in the OODB so that we can reuse them to acquire new knowledge. A method for efficient query evaluation is also required.

Acknowledgement

This work was supported in part by a Grant-in-Aid (04261101) for Scientific Research on Priority Areas from The Ministry of Education, Science and Culture in Japan.
References


