TERZO CONVEGNO NAZIONALE
SISTEMI EVOLUTI PER BASI DI DATI
Ravello, 28-30 Giugno 1995

Querying Genome Databases

Giansalvatore Mécca
Dipartimento di Informatica e Sistemistica
Universitá di Roma "La Sapienza"
Via Salaria, 113 – 00198 Roma, Italy
tel: +6 8558-418 - fax: +6 8530-0849
mecca@infokit.dis.uniroma1.it

Abstract

Genome Databases *contain genetic information about human beings and other species of living organisms. Since 1986, year in which the* Human Genome Project *started, they have grown exponentially in size every year. Due to the unusual structure of the stored information, traditional DBMS and query languages have provided only little support so far. In this paper we discuss some of the issues related to genomic information management. We mainly focus on query languages and present a logic, called* Sequence Datalog, *designed to be an effective tool for querying genome data.*

1 Introduction

In the last ten years, a bunch of new applications (e.g. CAD and CASE), have represented a powerful driving force towards database technology change. New paradigms have arisen (e.g. object oriented databases [2]), providing better flexibility in data management and query languages with respect to the traditional relational model. In this paper we discuss an example of these applications, namely the ones connected with the *Human Genome Project*, which are pointing out further interesting directions in database research and development.

The *Human Genome Project* is a research effort started back in 1986 as an initiative of two U.S. institutions, the *Department of Energy (DOE)* and the *National Institute of Health (NIH)*. The focus of the project is the study of genetic information, which represents the base for the life of every living organism. The purpose is to gain a better understanding of the nature of genetic and cellular mechanisms, that will provide an unvaluable source of information for medicine and health care in the next years.

Computer science and especially database technology has been by the side of genetics and molecular biology from the start. In fact, the huge amount of experimental data produced and the great complexity of experimental protocols require sophisticated computing applications to manage and interpret data. The main feature of these applications is represented on one side by the sequential structure of data to be stored, and on the other side by the complex nature of the transformations required to study these data. It is rather apparent the fact that commercial DBMS based on relational technology are not completely suited to this purpose. In fact, commercial products have been rather unsuccessful in this field so far.

This paper tries to sketch some of the issues related to sequential data management in the context of Genome databases, pointing out where the main weaknesses of current technology are. The paper mainly focuses on aspects related to query languages. In this context, a query language explicitly designed for handling genome sequences, called *Sequence Datalog* [15] is presented, and some interesting examples of queries are shown. Many of these examples are derived from our experience with the *Sequence Datalog* prototype, which is currently under development at "La Sapienza" University in Rome.

Due to the interdisciplinary nature of the matter, the paper is organized as follows. In Section 2 we introduce a few basic notions of molecular biology, necessary to gain a better understanding of the domain of interest and of examples throughout the paper. For a detailed introduction to the subject, we refer the reader to [14, 7]. Then, the Human Genome Project is briefly presented in Section 3. Technological issues related to Genome databases are discussed in Section 4, whereas Section 5 specifically focuses on aspects related to querying genomic data. Finally, the *Sequence Datalog* query language is presented in Section 6 and some examples are given in Section 7.

2 Notions of Molecular Biology

The term *genome* is used to refer to the whole genetic information of an organism. This information, stored in *nucleic acids*, the *deoxiribonucleic acid (DNA)*, and the *ribonucleic acid (RNA)*, contains the master blueprint for all cellular structures and activities.

Nucleic acids are complex molecules. Their constituents, called *nucleotides* or *bases*, are simpler molecules that we can divide in two groups: *(i) purines*, that is, *adenine*, usually denoted `a`, and *citosine* (`c`); *(ii) pyrimidines*, that is, *guanine* (`g`) and *timine* (`t`). Nucleotides bind to form long strands called *polynucleotides*, which are essentially polymers [1] in which nucleotides are linked to each other. A DNA molecule is made of two strands wrapped around each other to form a *double helix*. The double helix spatial configuration is due to weak hydrogenous bonds that are established between nucleotides on the two strands. This bonds, often referred to as *base pairings*, respond to a precise *complementarity* relationship according to which `a` only pairs with `t` and `c` only pairs with `g`. The bond is directional, that is, the DNA molecule has a head (usually called the $5'$ end) and a tail (called the $3'$ end), and one of the strands is called the *direct strand* and the other the *complementary strand.*

Example 1 The term *DNA sequence* denotes the arrangement of nucleotides in each of the two strands of a DNA molecule. Such sequences are naturally represented as

[1]A *polymer* is a long sequence of similar bonded elements.

strings over the alphabet $\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$.

For example, the following is a representation of a DNA fragment.

```
...  a c t t g c c a a ...
 5'  | | | | | | | | |  3'
...  t g a a c g g t t ...
```

□

The high grade of redundancy, which makes possible to produce high-fidelity copies of DNA molecules, is the very base of the replication process occurring in living cells.

In fact, the genetic information of an organism is stored in one or more long DNA molecules, called *chromosomes*. For example, in human beings, each cell contains in its nucleus 23 pairs of chromosomes. As long as the organism grows, that is, its cells duplicates, this information is passed on to new cells. In particular, whenever a cell duplicates, each of the *daughter cells* receives a copy of the chromosome set of the original cell. In order to do this, the cell first replicates its whole genome. During such replication, hydrogen bonds in DNA molecules break and strands separate; then, free nucleotides in the nucleus are used to build a new complementary strand for each of the original strands, thus originating two identical copies of the cell genome. In this way, each cell in the living organism has in its nucleus the whole genomic information, and can use it to perform its functions.

The means by which chromosomes affect the structure and the functions of living cells are *proteins*, that can be considered a primary component of living things. From the structural point of view, proteins are polymers themselves, made of basic constituents called *amino acids*. There are 20 different amino acids in nature. Amino acids form peptide bonds to create long chains called *polypeptides*; the sequence of amino acids that form a protein is called its *primary structure*, usually expressed as a string over a conventional 20-symbol alphabet $\{A, C, K, M, N, \ldots\}$.

Indeed, the primary role of nucleic acids is to carry the *encoding* of the structure of specific proteins. We call a *gene* each portion of chromosome containing information about the primary structure of a protein. It is worth noting that only about 10% of the whole human genome is known to contain protein-coding regions. The remaining part consists of noncoding regions – such as *control regions* – and other completely meaningless regions.

Roughly speaking, we can say that there is one gene for each specific protein in a living organism. There are, in fact, at least 100.000 different genes in humans. A simple and powerful code is used for storing such information. In particular, to each non-overlapping triplet of nucleotides in a gene, called a *codon*, corresponds a particular amino acid. Four distinguished codons are used as *stop codons*, that is, to signal the end of the coding sequence. It is easy to note that also in this case the code is redundant, since there are 4^3 different codons and only 20 amino acids (plus the *stop* signal).

When a protein is syntesyzed in a living cell, the DNA fragment representing the corresponding gene undergoes several transformations.

- the DNA sequence is first *transcribed* into an RNA sequence; RNA is a nucleic acid whose structure is very similar to the one of DNA; the only difference is that a new nucleotide, called *uracil* (u), takes the place of timine (t); RNA molecules are single-stranded, so that each RNA fragment can be easily represented as a string over the alphabet {a, c, g, u}; when the transcription of a DNA fragment occurs, the two strands separates and one of them is used as a template to build a RNA strand, according to a slightly different complementarity relationship (the only difference is that each a is transcribed to a u);
- then, the RNA molecule travels out of the nucleus, and undergoes a *splicing* process; in fact, genes are not necessarily contiguous inside chromosomes; if we consider the chromosome region containing a gene, there will be some gene regions that actually code for proteins, called *exons*, and some interleaving regions, called *introns*, which do not contain useful information; when DNA is expressed, introns need to be spliced out, so that exons can be connected together and the whole coding region can be processed; such splicing happens in specialized apparates called *spliceosomes*, which take a transcripted RNA sequence and splice out regions corresponding to introns, connecting exons together; the resulting molecule is called *messenger RNA*,
- the resulting RNA sequence is ready to be expressed; to do this, other apparates, called *transfer RNAs*, *translate* the sequence by simply reading the sequence codon by codon - that is, three symbols at a time - and, for each codon, appending the corresponding amino acid to the protein. For example, whenever the codon **aug** is encountered, the corresponding amino acid, *methionine*, is appended to the amino acid sequence. The four stop codons, namely **gct**, **gcc**, **gca** and **gcg**, signal that this translation process has to stop.

Example 2 As an example, consider the following DNA fragment representing a gene.

```
    1 2 3 4 5 6 7 8 9 0 1 2 3 4
    t a c g c c a a c t t a c t
 5' | | | | | | | | | | | | | | 3'
    a t g c g g t t g a a t g a
```

Suppose also that we know that the exons are from base 01 to base 06 and from base 09 to base 14. When the direct strand undergoes transcription, the following messenger RNA is produced.

```
a u g c g g u u g a a u g a
```

After splicing, the two exons are spliced together, so that the sequence to translate and the resulting protein are the following.

```
a u g c g g g a a u g a

<met> <arg> <glu> <stop>
```

Here, *met*, *arg* and *val* stand for the *methionine (M)*, *arginine (A)* and *glutamic acid (G)* aminoacids, respectively.□

3 The Human Genome Project

The final goal of the Human Genome project is to determine the exact sequence of the whole human genome, i.e. of every gene present in the human race, and the corresponding functions. To give a measure of the effort, it is worth noting that human genomes are about 3 billion base long. If compiled in books, the data would fill an estimated 200 volumes, each the size of a 1000 pages telephone book, and reading it would require about 30 years working around the clock.

The achievement of such an ambitious goal is unfortunately very far away for now, as the current experimental technology does not allow to read sequences longer than a few hundred bases. In fact, time and cost considerations make large-scale sequencing projects totally impractical. Consider that the smallest human chromosome (chromosome Y) is 50 million bases. The best available equipment can sequence only 50.000 to 100.000 bases per year at an approximate cost of $1 to $2 per base. At this rate, an incceptable amount of 30.000 work-years and at least $3 billion would be required for sequencing alone. It is apparent how the study of new sequencing technologies is a major concern in genome research these days.

Such a strong limitation imposes complex experimental protocols to reduce long chromosomes into manageable fragments. On the other side, the study of chromosomes is usually focused on different kinds of information rather than on the actual sequence. The objective is to build *maps* of chromosomes, that is, to find the approximate position of relevant sites on the chromosomes, corresponding to functionally important regions.

Due to the enormous size of human genomes and to the difficulty to perform experiments on human beings, much of the research work is performed on other species, as mice, insects, yeasts and bacteria. The following table gives an idea of the size of different genomes studied so far.

Comparative Sequence Sizes	
Yeast Chromosome 3	350.000 bases
Escherichia Coli (bacterium) genome	4.6 million bases
Entire yeast genome	15 million bases
Smallest human chromosome (Y)	50 million bases
Largest human chromosome (1)	250 million bases
Entire human genome	3 billion bases

4 Genome Databases

The huge amount of data generated by genome research will be used as a primary information source for human biology and medicine into the future. A main issue is therefore the management and distribution of genome data. Experimental information, sequences, maps need to be collected and stored in order to be fully accessible to the research community.

Clearly, a strong database technology is necessary to provide full support for these activities. Unfortunately, traditional DBMS have shown serious shortcomings in terms of genome data management [9].

The first category of limitations is concerned with the data model. In fact, DNA data requires a flexible *sequence data type* to be properly represented. Relational database systems do not provide such a type, so that the only way to implement a DNA sequence type is to store sequences as text strings. This technique, anyway, do not seem to be completely satisfactory, since, on one side a 8-bit per base representation is a clear waste of disk space - DNA sequences can be easily represented in a compress way by using only 2 bits per base; on the other side, manipulation of text fields, when present in commercial systems, is usually associated with serious limitations in terms of access. For example, indexing is usually not provided, and operators such as *substring* and *concatenation* need to be implemented by the user.

Indeed, these limitations have made the role of relational DBMSs a very poor one in genome research. As an example, consider the world most famous sequence database, *GenBank* [3]. *GenBank* is the NIH genetic sequence database, a collection of all known DNA sequences. There are approximately 230 million bases and 238.000 sequences as of December 1994. It represents a fundamental reference for the research community, since often sequenced DNA regions are required to be submitted to GenBank for electronic publication even before they are published in printed form. Information in GenBank is organized in records called *entries*; each *entry* contain a sequence along with some *annotations*, which provide details about the source organism, gene or DNA region, and the internal structure of the reported sequence, i.e. the number and position of exons, introns and relevant control regions.

Example 3 The following is an example of GenBank entry.

```
LOCUS       HUMNF1AB     8959 bp ss-mRNA      PRI       02-JAN-1992
DEFINITION  Human type 1 neurofibromatosis protein mRNA, complete cds.
ACCESSION   M82814
KEYWORDS    NF1 GAP-related protein; NF1 gene product;
            type 1 neurofibromatosis protein.
SOURCE      Homo sapiens cDNA to mRNA.
  ORGANISM  Homo sapiens
            Eukaryota; Animalia; Chordata; Vertebrata; Mammalia;
            Theria; Eutheria; Primates; Haplorhini; Catarrhini;
            Hominidae.
REFERENCE   1  (bases 1 to 8959)
  AUTHORS   Wallace,M.R., Marchuk,D.A., Andersen,L.B., Letcher,R.,
            Odeh,H.,Saulino,A.M., Fountain,J., Brereton,A.,
            Nicholson,J., Mitchell,A.,Brownstein,B.H. and Collins,F.S.
  TITLE     Type 1 Neurofibromatosis gene:
            identification of a large transcript
            disrupted in three NF1 patients
```

```
  JOURNAL   Science 249, 181-186 (1990)
  STANDARD  full automatic
REFERENCE   2  (bases 1 to 8959)
...
FEATURES    Location/Qualifiers
     CDS    212..8668
            /gene="NF1"
            /note="putative"
            /product="GAP-related protein"
            /codon_start=1
            /translation="MAAHRPVEWVQAVVSRFDEQLPIKTGQQNTHTKVSTEHNKECLI
            NISKYKFSLVISGLTTILKNVNNMRIFGEAAEKNLYLSQLIILDTLEKCLAGQPKDTM
                       ...
     source 1..8959
            /organism="Homo sapiens"
BASE COUNT  2598 a   2024 c   1888 g   2449 t
ORIGIN
   1 ccccagcctc cttgccaacg ccccctttcc ctctccccct cccgctcggc gctgaccccc
  61 catccccacc cccgtgggaa cactgggagc ctgcactcca cagaccctct ccttgcctct
 121 tccctcacct cagcctccgc tccccgccct cttcccggcc cagggcgccg gcccaccctt
     ....
7801 aaaggctcct aaaaggcaag aaatggaatc agggatcaca acaccccccca aaatgaggag
//
```

Entries are structured according to a fixed format, in which fields associated to sequences are used to *annotate* them. In this way, additional information about the sequence is stored in the database.□

Although GenBank uses a relational DBMS for internal data manipulation, external access to the database is not possible through a relational interface. In fact, the most common way to access the database is to download a flat file, text-based dump of the current version, in which all entries are listed sequentially.

Other very popular databases have also adopted alternative models of representation. For example, the *ACeDB* [19], the integrated *nematode worm* genome database, use a specialized hierarchical file system to represent information, and the user can navigate the database by following links from one record to the other. This model has proven very successful in the genome community, and other researchers are implementing databases for other species based on a similar architecture.

The lack of a unifying model for representing information is probably the first obstacle to be solved on the way of a real integrated access to genome data. In fact, the different databases spread in the world, containing informations about proteins, maps, metabolic pathways, speak different languages in terms of data models and data access facilities, so that today is not possible to ask for biologically relevant queries that involve data contained in different databases.

5 Query languages for genomic data

The other important cathegory of limitations of current database systems is related to query languages. The very rich structure of genomic sequences makes querying a critical activity. We can say that there are two different aspects of queries over sequences that have to be taken into account.

On one side, an expressive query language should provide powerful *pattern matching* capabilities. In fact, testing similarities of newly sequenced sequences against existing databases is a fundamental activity. Right now, a researcher willing to test the degree of omology of a new sequence against a database needs to recur to dedicated packages. In particular, there are some well known query servers, for example *BLAST* [1] or *FASTA* [16] which can be invoked through the Internet to test local alignment of short sequences against GenBank. Unfortunately, such specialized resources do not have a flexible query interface, so that, for example, one cannot specify queries such as *"test the similarities of sequence X against any sequence in the database in which the organism field equals* human", in which additional conditions are specified to narrow the range of interesting sequences and speed up the search.

To solve these problems, in the literature, several *pattern languages* have been proposed (e.g. [12, 18]). These languages usually provide a flexible syntax for defining *patterns*, that is, expressions that correspond to a set of strings over a given alphabet. In principle, one can think to enrich a relational query language, say SQL, by means of an extended *selection operator*, that, given a pattern, allows for extracting the set of strings in the database that match the pattern. Although the resulting language might represent a good solution to many sequence analysis problems, it would still be too weak in terms of *data restructurings*.

In fact, *restructurings* also play a central role. The raw sequence data usually provides little insight about the structure and functions of the corresponding DNA. For example, given a DNA sequence, one would like to be capable of finding all the regions of interest inside the sequence, that is, *(i)* exons; *(ii)* introns; *(iii)* important control regions responding to particular patterns or *consensus sequences*. More important, all these conceptually derived data should be accessible exactly as the base, stored data is, that is, the language should provide a flexible *view definition* mechanism to *restructure* information in the database. As an example, a simple way to represent exons and introns inside a given sequence is to identify them by means of the starting and ending position inside the sequence, but then, a primitive to retrieve the corresponding subsequence is necessary.

Moreover, sequences often need to be transformed. For example, it should be possible to find the reverse complement of a given sequence, or, given the set of exons of a gene inside a DNA sequence, splice exons together and translate them into the corresponding protein primary structure. This means that some primitives for computing sequence mappings are needed, potentially introducing new sequences in the database.

On the basis of the these considerations, we can briefly sketch a set of important requirements for a genome sequence query language as follows:

- the language should be *expressive*, that is, able to capture the complexity of genomic data, both in terms of pattern extraction and sequence restructurings;
- the language has to be declarative and easy to use; it should have a clear semantics and a nice syntax, and a suitable view definition mechanism, so that it can eventually become an effective tool for biologists;
- last but not least, the language should be *safe* in the sense that queries should always terminate, and the semantics be tractable; note, in fact, that queries on sequences may end up in nonterminating computations even when the base alphabet is finite; in fact, by growing in lenght, sequences can easily become infinite.

It is apparent how a good query language is the result of a good compromise between expressive power and effective computability. Various proposals have appeared in the recent literature about languages for querying sequences. Indeed, in most cases the trade-off between *expressiveness*, *finiteness* and effective *computability* is a hard one. In many cases [8], powerful logics for expressing sequence transformations are proposed, but a great part of the expressive power has to be sacrificed to enforce finiteness. In other cases [11], both expressiveness and finiteness are achieved, but at expense of an effective procedure for evaluating of queries, *i.e.*, at the expense of an operational semantics.

In the following section, we present a language for querying sequences, called *Sequence Datalog*, which meets most of the requirements listed at the beginning of the section, that is, it has a nice horn-logic like syntax, a clear declarative semantics, a simple operational semantics and allows for the definitions of sublanguages which are expressive, safe and tractable at the same time.

6 Sequence Datalog

In this section we refer to a data model which extends the relational model to embed sequences. Our databases are collections of tables; each table is a set of records containing typed fields. We essentially allow two types of data:

- traditional *atomic* data, essentially fixed lenght strings and integers;
- *sequences*, that is, textual information of unbounded lenght usually stored in files; these fields are used to store genomic sequences.

The *Sequence Datalog* language [15] is a Datalog [4] like language for this extended relational model. The semantics of Sequence Datalog programs is similar to the one of Datalog. Programs are set of rules of the form:

$$p_0(\ldots) \leftarrow p_1(\ldots), \ldots, p_n(\ldots).$$

where p_0 is a *derived predicate*, and each of the $p_i, i = 1, \ldots, n$ in the body is either a base predicate corresponding to a database relation or a derived predicate itself.

With respect to Datalog, two special, interpreted function symbols are introduced to manipulate sequences. More specifically, *Sequence Datalog* programs use *indexed terms* to extract subsequences and *constructive terms* to concatenate them. Any *indexed term* of the form $s[n_1{:}n_2]$ is interpreted as a *subsequence* of s, whereas a *constructive term* of the form $s_1 \bullet s_2$ is interpreted as the *concatenation* of s_1 and s_2, that is, conceptually, a *supersequence* of both.

So, for example, the following rule extracts all the prefixes of the sequences in relation R_1

$$prefix(X[1{:}N]) \quad \leftarrow \quad R_1(X).$$

This rule specifies that, for every sequence X in R_1, a prefix of X is any subsequence starting with the first element and ending with the N-th element, as long as N is not greater than the length of X.

Since the universe of strings over a given alphabet, Σ^*, is an infinite set, in order to keep the semantics of programs finite we do not evaluate rules over the whole universe Σ^*. In fact, we introduce a new active domain for sequence databases, called the *extended active domain*; this domain contains all the sequences occurring in the database, plus all their *subsequences*. [2] Then, we require that substitutions range over this domain when rules are evaluated. [3]

This extended active domain is not fixed during the evaluation of a query. In fact, it can grow, since new sequences can be added by means of *constructive terms*, which concatenate existing sequences to produce new ones. When this happens, the new sequence – and its subsequences – become part of the extended domain. Note that because *Sequence Datalog* is a Horn-like logic, we can give a declarative meaning based on fixpoint theory to this apparently procedural notion.

For example, the following query constructs all the possible concatenations of sequences in relation r_1 and adds them to the *extended active domain* of the database;

$$answer(X \bullet Y) \quad \leftarrow \quad R_1(X), R_1(Y).$$

The rule takes any pair of sequences X and Y in relation R_1, concatenates them and stores the result in *answer*, potentially introducing new sequences into the *extended active domain* of the input database.

Compared to Datalog with function symbols or Prolog, two differences are apparent. The first is that noninterpreted functors are not allowed in Sequence Datalog, so that it is not possible to build arbitrarily nested structures. On the other hand, Sequence Datalog has a richer syntax than the list constructor $[Head|Tail]$ of Prolog. This has a great impact on the way *structural recursion* is handled in Sequence Datalog. In fact, Sequence Datalog distinguishes *syntactically* between structural recursion

[2]In this paper, we always refer to *contiguous subsequences*, that is, subsequences of a given sequence identified by a start and an end position in the sequence. So, for example, *bcd* is a contiguous subsequence of *abcde*, whereas *bd* is not.

[3]Note that the the *extended domain* has at most quadratic size with respect to the size of the domain of the database. In fact, the number of different contiguous subsequences of a given sequence of length k is $\sum_{i=0}^{2} \binom{k}{i}$, that is, $\frac{k(k+1)}{2} + 1$.

inside existing sequences and recursion through *construction of new sequences*, whereas typically, languages for list manipulations do not discriminate the two types. In fact, in *Sequence Datalog*, structural recursion over existing sequences – which is inherently safe, since it does not make the active domain grow – is handled by means of *indexed terms* of the form $X[n_1{:}n_2]$, whereas recursion through construction of new sequences is performed with *constructive terms* of the form $X \bullet Y$, which can be unsafe, since it introduces new elements in the domain.

Example 4 As an example, suppose we are interested in sequences that are multiple repeats – that is, contain one or more copies – of other sequences, as in *abcdabcdabcd*. Such repeats are patterns of great importance in Genome databases. There are two straightforward ways of expressing this query in *Sequence Datalog*. Both involve structural recursion, but their semantics is very different.

$$\begin{array}{lll} rep_1(X,X) & \leftarrow & true. \\ rep_1(X,X[1{:}N])) & \leftarrow & rep_1(X[N+1{:}end],X[1{:}N]). \\ \\ rep_2(X,X) & \leftarrow & true. \\ rep_2(X \bullet Y,X) & \leftarrow & rep_2(Y,X). \end{array}$$

When these rules are evaluated over an input database, predicate rep_1 looks for multiple repeats "inside" *existing sequences*, that is, sequences in the active domain of the input database; it is easy to see that, even though in the answer there can be sequences that are not in the database, they will be subsequences of existing sequences, so that the semantics is finite over every database. We call this *safe structural recursion*. In contrast, predicate rep_2 "builds up" new sequences by recursively concatenating each sequence in the database with itself; in this way, all possible multiple repeats of sequences in the given database are generated, causing a nonterminating computation, unless the database is empty.□

In general, predicates such as rep_2, that are recursive through constructions of new sequences are potentially *unsafe*, that is, can produce infinite answers. Since we are only interested in finite answers, that is, in safe programs, we enforce termination by carefully limiting the amount of recursion through constructions in our programs. In order to do this, we use safe recursion, that is, recursion over sequences in the domain, to control unsafe recursion, as in the following example.

Example 5 The following rules compute the reverse complement of each sequence in relation R_1, supposing that relation *compl* encodes the complementarity relation between nucleotides.

$$\begin{array}{lll} answer(X,Y) & \leftarrow & R_1(X), rev\_complement(X,Y). \\ rev\_complement(\epsilon,\epsilon) & \leftarrow & true. \\ rev\_complement(X, Z \bullet Y) & \leftarrow & compl(X[1],Y), \\ & & rev\_complement(X[2{:}end],Z). \end{array}$$

The rules constructs the reverse complement of each sequence by appending the complementary base for each base in the original sequence, in reverse order. Note that predicate *rev_complement* is recursive through constructions. Though, the set of answers produced is finite over every database. In fact, safe recursion over the first attribute is used to control the generation of new sequences in the second attribute, so that it is not possible to construct new sequences that are longer than the longest sequence in the initial active domain: by bounding above the lenght of the constructed sequences, termination is guaranteed.

Predicate *rev_complement* can be seen as if encoding the computation of an abstract machine – a *transducer*, essentially – that takes a sequence as input and produces its reverse complement as output. The computation of this machine is controlled by the lenght of its input, so that it always terminate for finite lenght inputs. □

These considerations allow for a straightforward definition of *finite subsets* of the logic, that is, restrictions that guarantee termination. In [15], two different sublanguages of Sequence Datalog are defined and their expressive power is established. The first one expresses exactly the class of PTIME sequence functions. This language, which provides an interesting charachterization of PTIME, represents a good compromise between expressive power and a feasible semantics. The second language, on the other side, can expess any sequence mapping with hyper-exponential time complexity.

7 Querying genome databases

Sequence Datalog represents a flexible tool for querying Genome databases, as shown in the following examples. In fact, the language has both powerful pattern matching and restructuring capabilities. In the following, we discuss both aspects and give some interesting examples of Sequence Datalog queries.

7.1 Pattern matching

Roughly speaking, we can define a *pattern* as a set of sequences sharing some structural similarity. Patterns are usually defined by means of *formal grammars* [13]. The activity of recognizing a pattern is thus reduced to parsing the corresponding grammar. Sequence Datalog allows for a simple parsing of *context free languages* [13], as shown in the following example.

Example 6 Genes, that is, DNA coding regions, are usually preceeded by an *upstream region*, which contains important signals, called *promoters*, which control gene expression. The following is a simple context-free grammar that reproduces the structure of these promoter regions, based on some important patterns such as *caat boxes* and *tata boxes*. In the following, $\mathtt{a}, \mathtt{t}, \mathtt{c}, \mathtt{g}$ are terminals, whereas nonterminals are written in the form $\langle nonterminal \rangle$. The start symbol of the grammar is $\langle upstream \rangle$.

$$\begin{array}{rcl}
\langle upstream \rangle & \rightarrow & \langle caatbox \rangle \langle gap \rangle \langle tatabox \rangle \\
\langle tatabox \rangle & \rightarrow & \mathtt{tata}\langle base \rangle \mathtt{a} \\
\langle caatbox \rangle & \rightarrow & \langle base \rangle \mathtt{caat} \\
\langle base \rangle & \rightarrow & \mathtt{a} \\
\langle base \rangle & \rightarrow & \mathtt{g} \\
\langle base \rangle & \rightarrow & \mathtt{c} \\
\langle base \rangle & \rightarrow & \mathtt{t} \\
\langle gap \rangle & \rightarrow & \langle base \rangle \langle gap \rangle \\
\langle gap \rangle & \rightarrow & \epsilon
\end{array}$$

This grammar can be used to parse DNA sequences in order to recognize upstream regions. Given a database relation DNA_*sequence*(*id*, *sequence*), the following Sequence Datalog program parses every sequence and returns upstream regions in predicate *upstream_region*; the idea of the program is to associate a predicate to each nonterminal in the grammar, as follows.

$$\begin{array}{lll}
upstream\_region(X) & \leftarrow & \mathrm{DNA}\_sequence(X, Y), upstream(Y). \\
upstream(X) & \leftarrow & caatbox(X[1{:}N_1]), \\
 & & gap(X[N_1+1{:}N_2]), \\
 & & tatabox(X[N_2+1{:}end]). \\
tatabox(X) & \leftarrow & X[1{:}4] = \mathtt{tata}, \\
 & & base(X[5{:}end-1]), \\
 & & X[end] = \mathtt{a}. \\
caatbox(X) & \leftarrow & base(X[1{:}end-4]), \\
 & & X[end-3{:}end] = \mathtt{caat}. \\
base(X) & \leftarrow & X = \mathtt{a}. \\
base(X) & \leftarrow & X = \mathtt{t}. \\
base(X) & \leftarrow & X = \mathtt{c}. \\
base(X) & \leftarrow & X = \mathtt{g}. \\
gap(\epsilon) & \leftarrow & true. \\
gap(X) & \leftarrow & base(X[1]), \\
 & & gap(X[2{:}end].
\end{array}$$

The rules of the program are derived from the grammar productions in a straightforward way. It is easy to prove that the program correctly recognizes the language associated with the grammar. □

Note that GenBank entries are structured according to a context-free grammar, so that parsing can be easily used to extract information from the flat-file text dump and introduce it into a custom database.

Although other list-based languages as Prolog [5] also parse context free languages, Sequence Datalog allows for more powerful parsing that goes beyond context-free languages. In fact, it would be possible to prove that any context-sensitive language can be parsed in Sequence Datalog. We show this by means of an example.

Example 7 Suppose we are interested in all sequences of the form $a^n b^n c^n, n \geq 0$ in relation r_1; these patterns are known to be beyond context free languages, and usually a complex context sensitive grammar would be necessary for the parsing. Nevertheless, the query can be expressed with a simple Sequence Datalog program, as follows.

$$\begin{array}{lcl}
answer(X) & \leftarrow & R_1(X), \\
 & & abc_n(X[1{:}N_1], X[N_1+1{:}N_2], X[N_2+1{:}end]). \\
abc_n(\epsilon,\epsilon,\epsilon) & \leftarrow & true. \\
abc_n(X,Y,Z) & \leftarrow & X[1]=a, Y[1]=b, Z[1]=c, \\
 & & abc_n(X[2{:}end], Y[2{:}end], Z[2{:}end]).
\end{array}$$

Predicate abc_n is true for every triple of sequences of the form (a^n, b^n, c^n) in the *extended active domain* of the database; $answer(X)$ is true for a sequence X in R_1 if it is possible to split X in three parts such that abc_n is true.□

These flexible parsing capabilities are very important in the context of Genome databases, since it has been proven [6] that not all the pattern of interest in a DNA molecule are context-free. The main example are *multiple repeats* in DNA sequences (see Example 4), which are have important biological functions and are not a context-free language. It is worth noting that this represents a true peculiarity of genomic data, since in other fields context-free languages are considered an acceptable compromise. Since context-sensitive grammars are too complex and parsing is rather inefficient, the computational linguistic of DNA has become a field of great interest, and several models of grammars that stand in between context-free and context-sensitive languages have been defined [17, 18]. The definition of these models is beyond the scope of this paper. Anyway, Sequence Datalog allows for a simple parsing of these grammars as well.

7.2 Data restructurings

The transducer-like model of computation introduced in section 6 has great expressive power in terms of sequence restructurings. In particular, all the transformations over sequences produced in the cell can be easily modelled. In fact, cell apparates are nothing more than very sophisticated *biological transducers*, that take a sequence as input and transform it into an output sequence.

Example 8 Suppose that our database contains the following predicates:

- DNA_*sequence*(*id*, *sequence*, *length*), containing information about a set of DNA sequences, each with a numeric identifier;
- *exon*($sequence_{id}$, *start*, *end*) and *intron*($sequence_{id}$, *start*, *end*), that specify the starting and ending positions of each exon and intron contained in a sequence in relation DNA_*sequence*.

In this case, the splice of a given sequence can be derived by means of the following query;

$$\begin{array}{lllcl}
\gamma_0: & splice(X,Y) & & \leftarrow & \mathrm{DNA}\_sequence(X,S,L), \\
 & & & & splicing(X,Y,L). \\
\gamma_1: & splicing(X,\epsilon,0) & & \leftarrow & \mathrm{DNA}\_sequence(X,S). \\
\gamma_2: & splicing(X,Y,N_2) & & \leftarrow & \mathrm{DNA}\_sequence(X,S), \\
 & & & & splicing(X,Y,N_1), \\
 & & & & intron(X,N_1+1,N_2). \\
\gamma_3: & splicing(X,Y \bullet S[N_1+1{:}N_2],N_2) & & \leftarrow & \mathrm{DNA}\_sequence(X,S), \\
 & & & & splice(X,Y,N_1), \\
 & & & & exon(X,N_1+1,N_2).
\end{array}$$

Using this *splice* predicate, we can define the *protein* predicate, which associates the corresponding protein with each DNA sequence in the database. We suppose that predicate *rna_compl* encodes the complementarity relationship between DNA and RNA nucleotides, and that predicate *codon* associates an amino acid with each codon.

$$\begin{array}{llcl}
\gamma_4: & protein(X,P) & \leftarrow & \mathrm{DNA}\_sequence(X,\mathrm{DNA}), \\
 & & & trascribe(\mathrm{DNA},\mathrm{MRNA}), \\
 & & & splice(\mathrm{MRNA},\mathrm{RNA}), \\
 & & & translate(\mathrm{RNA},P). \\
\gamma_5: & transcribe(\epsilon,\epsilon). & \leftarrow & true. \\
\gamma_6: & transcribe(X,Z \bullet Y) & \leftarrow & rna\_compl(X[1],Z), \\
 & & & transcribe(X[2{:}end],Y). \\
\gamma_7: & translate(\epsilon,\epsilon). & \leftarrow & true. \\
\gamma_8: & translate(X,Z \bullet Y) & \leftarrow & codon(X[1{:}3],Z), \\
 & & & translate(X[4{:}end],Y).
\end{array}$$

It is easy to see how each DNA sequence is first transcribed into an RNA molecule, then exons are spliced out and the resulting coding sequence is translated into the corresponding protein.□

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman. Basic Local Alignment Search Tool. *Jour. of Mol. Biology*, 215:403–410, 1990.

[2] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and Z. Zdonik. The object oriented database manifesto. In *1st Intl. Conference on Deductive and Object Oriented Databases (DOOD)*, Kyoto, 1989.

[3] C. Burks, M. Cassidy, M. J. Cinkosky, K. E. Cumella, P. Gilna, J. Hayden, G. M. Keen, T. A. Kelley, M. Kelly, D. Kristofferson and J. Ryals. GenBank. *Nucleic Acids Res.*, 19*(Suppl.)*:2221–2225, 1991.

[4] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Data Bases.* Springer-Verlag, 1989.

[5] W. Clocksin, C. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.

[6] J. Collado Vides. The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context–free grammars. *Computer Appications in the Biosciences*, 7(3), 1991.

[7] Department of Energy. *Primer on Molecular Genetics*, 1994.

[8] S. Ginsburg and X. Wang. Pattern matching by RS-Operations: Towards a unified approach to querying sequenced data. In *11th ACM SIGMOD Symp. on Principles of Database Systems (PODS)*, 1992.

[9] N. Goodman, S. Rozen and L. Stein. A glimpse at the DBMS Challenges Posed by the Human Genome Project. Whitehead Institute for Biomedical Research, 1994.

[10] N. Goodman, S. Rozen and L. Stein. Requirements for a Deductive Query Language in the MapBase Genome Mapping Database. In *Workshop on Programming with Logic Databases (following ILPS '93)*, 1993.

[11] G. Grahne, M. Nykanen and E. Ukkonen. Reasoning about strings in databases. In *13th ACM SIGMOD Symp. on Principles of Database Systems (PODS)*, 1994.

[12] C. Hegelsen and P. R. Sibbald. PALM – A pattern language for molecular biology. In *Proc. First Int. Conference on Intelligent Systems for Molecular Biology*, 1993.

[13] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley, 1979.

[14] L. Hunter. Molecular Biology for computer scientists. In L.Hunter, editor. *Artificial Intelligence and Molecular Biology*, AAAI Press, 1994.

[15] G. Mecca and A. J. Bonner. Sequences, Datalog and Transducers. To appear in *14th ACM SIGMOD Symp. on Principles of Database Systems (PODS)*, 1995.

[16] W. B. Pearson and D. Lipman. Improved tools for biological sequence comparison. In *Proc. Nat. Academy of Science*, 85(8):2444–2448, 1988.

[17] D. B. Searls. Representing genetic information with formal grammars. In *Proc. Nat. Conf. of the American Association for Artificial Intelligence*, 7:386–391, 1988.

[18] D. B. Searls. Investigating the linguistic of DNA with Definite Clause Grammars. In *Proc. North Am. Conference on Logic Programming*, pages 189–208, 1989.

[19] J. Tierry-Mieg and R. Durbin. ACeDB: A database for genomic information. In *Proceedings of the Genome Mapping and Sequencing Workshop*, Cold Spring Harbor Laboratory, New York, 1992.