A Primer on Rapid Prototyping
of Genomic Databases in Prolog

Revised by STI
MAR 05 1992

by

Kaoru Yoshida, Cassandra L. Smith, and Ross Overbeek

January 1992

**MATHEMATICS AND
COMPUTER SCIENCE
DIVISION**

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States government, and operated by The University of Chicago under the provisions of a contract with the Department of Energy.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Reproduced from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62

Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM--160
DE92 009858

ANL/MCS-TM-160

A Primer on Rapid Prototyping of Genomic Databases in Prolog

by

Kaoru Yoshida,* Cassandra L. Smith,* and Ross Overbeek

Mathematics and Computer Science Division

Technical Memorandum No. 160

January 1992

*Department of Molecular Biology, University of California, and Division of Chemical Biodynamics,
Lawrence Berkeley Laboratory, Berkeley, California

This work was supported in part by DOE Grant No. DEFG02-87 and DOE Contract W-31-109-Eng-38.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Contents

Abstract	1
1 Introduction	1
2 Representing a Physical Map	2
3 Representing a Genetic Map	5
4 Representing Sequence Fragments	6
5 Representation of Structural Genes	7
6 Adopting an “Object-oriented” View	7
6.1 Lifting Facts to Terms	7
6.2 Providing an “Object-oriented” Interface	8
7 Summary	9
References	9

A Primer on Rapid Prototyping of Genomic Databases in Prolog

Kaoru Yoshida, Cassandra L. Smith, and Ross Overbeek

Abstract

This report presents a tutorial on how one might create an integrated database of genomic information. We outline the required steps for implementation, give a brief introduction to Prolog, and discuss the query facility supported by our system. Our goal is to enable researchers to be constructing their own biological information system.

1 Introduction

The development of new technologies and experimental approaches allows an ever-increasing amount of genomic information to be accumulated daily. Thus, the need for integrated databases to support genomic research grows more apparent. Recently, we participated in the design and implementation of two genomic database systems based on a logic programming language, Prolog. One system focused on the *Escherichia coli* genome [2], and another system focused on human chromosome 21 [11]. The available information on these two chromosomes is quite different and represents extremes of information content. For instance, approximately one third of the *E. coli* DNA sequence is available in Genbank and EMBL. Furthermore, several major efforts are under way to complete the sequence of the entire *E. coli* genome. Hence, it is likely that this genome will be one of the first to be completely sequenced.

Several groups (see [6, 8]) have developed relational databases for *E. coli* focusing on DNA sequence information and the genomic high-resolution restriction map constructed by Kohara [5]. The Prolog implementation project we participated in also focused on the Kohara data. The information integrated in the database includes the data made available by Rudd [8] and the *E. coli* genetic map created by Bachmann [1]. Rudd's data included an adjusted version of Kohara's physical map, melded DNA sequence data, and the positions of structural genes within the sequence melds.

The rapid prototyping ability of Prolog allowed us to test particular designs in a short period of time. Specifically, we developed a prototype of a functional system, allowing a limited set of users to experiment with the system, determined its deficiencies, and developed a new prototype. This cycle is practical only when tools like Prolog reduce the overall effort of developing such prototypes.

The system described here supported a querying facility not only in Prolog but also in a "pidgin" English dialect. The latter should eventually allow the relatively naive user flexible access to the data [3].

A complete, detailed guide to the design and implementation of integrated data resources is necessarily lengthy. This report offers a brief tutorial on how one might begin to integrate genomic information. The user will need a computational environment that supports both a decent text editor and an implementation of Prolog. One could use an IBM-PC, a Macintosh, or a Unix workstation. Our implementations have used Quintus Prolog on standard Unix workstations. While a smaller platform would be adequate for text editing, data validation, and some of the query processing, the available Prolog systems on larger workstations are far more robust.

Our goal is to give readers a flavor of Prolog and—for those willing to invest a few hours in considering the issues raised here—a base to begin constructing their own biological information system. By encouraging biologists to undertake this task, we hope to "demystify" the process and establish a framework to support future genomic analysis.

2 Representing a Physical Map

Complete restriction maps and libraries of overlapping clones do not exist for most chromosomes. Hence, if we are to implement a database capable of storing information, we shall need the capability of describing partially completed maps.

One way to represent such data is based on the notion of a set of "islands" that have been mapped. These islands may or may not yet have determined positions on the chromosome. As the map is completed, it will reach the point where a single island exists that covers the entire chromosome. However, throughout the entire process of mapping the chromosome, sections of known sequence may start to appear and to be assigned locations on islands. When an island has been positioned on the chromosome, the sequences positioned on the island will have been assigned a position on the chromosome; otherwise, the island, along with the attached sequence fragments, will be thought of as "floating." In the case of physical maps, the notion of island coincides with what is generally called "contigs." However, we shall use the term island in a more general sense, in that one may speak of an island for which only genetic maps exist (i.e., prior to the point where a physical map has been created).

Suppose that we have a set of clones that have been used to create a partial physical map of some specified chromosome. To be concrete, we shall speak of chromosome 1 of the fictitious organism *P. ins*. Then, one can naturally express information about where islands and clones occur on the chromosome with simple tables. For example, information about the overlapping clones in a map might be represented in tabular form as shown in Table 1.

Table 1: Clones for *P. ins*

Id	Begin	Length	Island
[101]	0	16870	1
[102]	9400	14757	1
[103]	20930	17570	1
[104]	27600	14300	1
:	:	:	:
[174]	0	20400	2
[175]	12600	18600	2
[176]	18350	13800	2
:	:	:	:

Essentially each clone has a unique identification, a start point in base pairs (bp) relative to the start of an island, and a size in bp. Islands may have a position relative to the start of the chromosome, or the position may still be "unknown" (see Table 2).

Table 2: Islands for *P. ins*

Id	Begin	Chromosome
1	383	1
2	unknown	1
:	:	:

Restriction site information can likewise be represented in tabular form, as shown in Table 3.

Here the location of restriction enzyme cutting sites are assigned to particular bp on an object, where the object can be the chromosome, an island, or a designated clone.

How can the data in such tables, which is in an easily understood form, be converted into a form suitable for a Prolog database? Three files should be created, one for each of the tables. First, create a file called "clones" containing the information contained in Table 1 (a listing of all the clones in the overlapping library).

Table 3: Restriction Sites for *P. ins*

Begin	Enzyme	On
10600	BglI	chromosome 1
11458	BglI	chromosome 1
26110	PvuII	chromosome 1
:	:	:
600	BglI	island 2
4670	EcoRV	island 2
4692	HindIII	island 2
:	:	:
102	HindIII	clone [174]
460	BglI	clone [174]
:	:	:

Then, use a text editor to convert all the lines in the table containing the raw data into the form required by Prolog. For example, suppose we choose to represent an entry in the clone table using a fact of the form

```
clone(Id, Genome, Island, Position, Length).
```

Then, one might represent the first line of the table as

```
clone('101', 'P.ins', 1, 0, 16870).
```

Such a file is thought of as a file of "facts." For example, the above fact is the assertion that

"Clone [101] occurs on the *P.ins* genome on island 1, starting at location 0 bp, and has a length of 16870 bp."

Similarly, in a second file, called "islands," one might assert facts of the form

```
island(Id, Genome, Chromosome, Position).
```

For example,

```
island(1, 'P.ins', 1, 383).
```

would be thought of as asserting

"Island 1 occurs on the *P.ins* genome on chromosome 1, starting at location 383 bp."

In the third file, called "rsites," the contents of Table 3 should be converted to Prolog facts in the following form:

```
rsite(Genome, Object, Location, Enzyme).
```

where "Object" indicates which object the restriction site occurs on. For example,

```
rsite('P.ins', chr(1), 10600, 'BglI').
rsite('P.ins', island(2), 600, 'BglI').
rsite('P.ins', clone('174'), 102, 'HindIII').
```

would be encodings of some of the selected entries from the table. Here, the assertions are

"The *P.ins* genome has an occurrence of a BglI site at location 10600 bp on chromosome 1, a BglI site at location 600 of island 2, and a HindIII site at location 102 of clone [174]."

Once these files have been created, they can be immediately queried using Prolog. For example, consider the following "session." The user first enters Prolog and types in

```
?- [clones, islands, rsites].
```

which causes Prolog to "consult" three files (clone, islands, and rsites). The Prolog program then carries out three queries. Here is the first.

query 1

```
?- clone(Id, 'P.ins', 1, Begin, Length),  
    Begin =< 25360, 25360 < Begin+Length.
```

```
Begin = 20930,  
Id = '[103]',  
Length = 17570;
```

```
no.
```

Query 1 involves the satisfaction of three goals:

1. clone(Id, 'Pins', 1, Begin, Length).
2. Begin =< 25360
3. 25360 < Begin+Length

To satisfy these three goals, Prolog must find values for the variables Id, Begin, and Length such that three goals are "true." That is, the system is asked to find a clone that begins at a position equal to or less than 25360 bp (in island 1 of the genome for *P.ins*) and extends beyond 25360 bp. Prolog will find values that satisfy the statements and will then wait for the user to decide whether more suitable values are desired. When the user responds with a ";", Prolog will search and report the next alternative, if there is one. Note that "variables" whose values are to be set by Prolog are denoted as words that begin with uppercase letters. In contrast, constants begin with lowercase letters or are enclosed within quotation marks.

Query 2 asks for the location of BglI sites that have a known position on chromosome 1 of the genome of *P. ins*. After two sites are identified, no more are asked for.

query 2

```
?- rsite('P.ins', chr(1), Loc, 'BglI').
```

```
Loc = 10600;  
Loc = 26110
```

Query 3 requests Prolog to find every occurrence of a BglI site that has a known position on the chromosome, and to collect all the occurrences into AllOccs in the form of a list.

query 3

```
?- setof(Occ, rsite('P.ins', chr(1), Occ, 'BglI'), AllOccs).
```

```
AllOccs = [10600,11458,...]
```

```
?- setof(Occ, rsite('P.ins', chr(1), Occ, 'BglI'), AllOccs).
```

We realize that the details of encoding data into Prolog and issuing such queries will require some careful study by the reader. From this brief discussion, however, one can appreciate our main point: that tabular information can be immediately converted to files of facts, and then queried with no additional effort.

3 Representing a Genetic Map

Genetic maps are used to represent approximate locations of genes within an island (once the entire chromosome has been mapped, this amounts to a situation in which a single island spans the entire chromosome). Information in an entry of a genetic map usually includes the gene name, genetic map position, and direction of transcription. The position will be expressed in any of several types of units, depending on the map (e.g., minutes or centiMorgans). This can also be represented in tabular form, as shown in Table 4.

Table 4: Genetic Map

Gene	Island	Position	Direction
txrA	1	0	unknown
txrB	1	0.1	unknown
rvgV	1	0.4	backward
:	:	:	:
trpR	2	0.6	unknown
pxoM	2	1.8	forward
:	:	:	:

The data from Table 4 could be encoded as Prolog facts of the form:

```
mapped_gene(Id, Genome, MapId, Island, Position).

mapped_gene_direction(Id, Genome, MapId, Direction).
```

For example,

```
mapped_gene(txrA, 'P.ins', 'Klagle', 1, 0).
mapped_gene(txrB, 'P.ins', 'Klagle', 1, 0.1).
mapped_gene(rvgV, 'P.ins', 'Klagle', 1, 0.4).
mapped_gene(trpR, 'P.ins', 'Klagle', 1, 0.6).
mapped_gene(pxom, 'P.ins', 'Klagle', 1, 1.8).

mapped_gene_direction(rvgV, 'P.ins', 'Klagle', backward).
mapped_gene_direction(pxom, 'P.ins', 'Klagle', forward).
```

Here, we have used a distinct type of assertion to specify the direction of transcription, when it is known.

The file of these facts can now be queried. However, some questions requiring integrating the physical and genetic map data cannot be asked directly, because the units of the maps are different (e.g., bp versus minutes, respectively). Integration of these two maps requires a rule to convert between the units. A simple rule for conversion might be

```
genetic_position_to_bp('P.ins', 'Klagle', GenPos, Bp) :-
    Bp is 36826 * GenPos.
```

This means: "For the *P.ins* genome, the bp position can be calculated from the genetic position, in the Klagle map, by multiplying by 36,826."

The following interaction can occur if the facts from Table 4 are put into a file called **gen_maps** and the rule about conversion into a file called **rules**

```
?- [gen_maps, rules].
?- clone('[113]', 'P.ins', 1, ClonePos, CloneLength),
    mapped_gene(GeneId, 'P.ins', MapId, 1, GeneLoc),
    genetic_pos_to_bp('P.ins', MapId, GeneLoc, GenePos),
```

```

ClonePos =< GenePos, GenePos < ClonePos+CloneLength.

CloneLength = 17083,
ClonePos = 119113,
GeneId = axeF,
GeneLoc = 2.696978,
GenePos = 126019;

CloneLength = 17083,,
ClonePos = 119113,
GeneId = ldpX,
GeneLoc = 2.744424,
GenePos = 128236;
:

```

We might also now add a rule as follows:

```

mapped_genes_in(Clone, Genome, Island, Genes) :-
    setof(Gene, gene_in_clone(Clone, Genome, Island, Gene), Genes).

gene_in_clone(CloneId, Genome, Island, Gene) :-
    clone(CloneId, Genome, Island, ClonePos, CloneLength),
    mapped_gene(Gene, Genome, MapId, Island, GeneLoc),
    genetic_position_to_bp(Genome, MapId, GeneLoc, GenePos),
    ClonePos =< GenePos, GenePos < ClonePos + CloneLength.

```

4 Representing Sequence Fragments

Eventually, the entire DNA sequence of a number of chromosomes will be known. Until that time, one must be able to represent the numerous islands of DNA sequence that are accumulating. Information about these sequence patches is accumulated in central repositories such as Genbank, EMBL, and the Japanese Databank. Specific sequences in these databases have been assigned unique identifications, but only some of these have been successfully placed on genomic restriction maps. This patchy sequence information can be represented by using two types of facts:

```

sequence_fragment(FragmentId, Genome, Sequence).

sequence_location(FragmentId, Genome, Island, Offset, Length).

```

For example,

```

sequence_fragment('ECOTHR', 'E.coli', "TATAGGCATAGCGCACAGA ...").
sequence_location('ECOTHR', 'E.coli', 1, 11, 5922).

```

would be used to assert the existence of the sequence fragment ECOTHR as a part of the *E. coli* genome, that it was positioned on island 1 at an offset of 11, and that it is 5922 bp in length.

A position can be asserted only when a fragment has been assigned a location on the genomic restriction map. In fact, the set of these facts may become quite large. Some Prolog implementations on small computers may not be able to handle a large collection of sequences, but this problem does not occur in better implementations on workstations. Furthermore, implementations on smaller machines usually have the ability to keep selected files of facts on disk, allowing an almost arbitrarily large collection of sequences.

5 Representation of Structural Genes

One of the central uses of a genomic database is to respond to queries about genes. Hence, it is critical that information relating to the position of genes be properly encoded. We propose the following, somewhat oversimplified approach.

Consider Tables 5 and 6: (Note that Begin in Table 5 is an offset from the first position of the sequence fragment.)

Table 5: Exons

Gene	Exon	Sequence Fragment	Begin	Length
thrA	1	ECOTHR	196	2643
thrB	1	ECOTHR	2660	930
thrC	1	ECOTHR	3590	1287

Table 6: Structural Genes

Id	Exons	Direction
thrA	1	forward
thrB	1	forward
thrC	1	forward

Although the translation of most structural genes that code for proteins can be computed directly from the sequence of the exons, this is not true in all cases; hence, we include a separate, explicit table (Table 7) giving the translation (when it is appropriate and available).

From the tables, we can immediately produce the encoded facts in the following form:

```
exon(thrA, 1, 'E.coli', 'ECOTHR', 196, 2643).
structural_gene(thrA, 'E.coli', 1, forward).
translation(thrA, "MRVLKFGGTSLANAERFLRVADILESNAQQG ...").
```

The encoding is minimal, in that nothing has been said about transcription, promoters, or other regulatory features. However, this minimal representation is extensible; that is, it is reasonably straightforward to add additional facts that assert the existence of such additional information when it is known.

6 Adopting an "Object-oriented" View

6.1 Lifting Facts to Terms

When designing a system that provides uniform manipulation of a set of objects, it is often convenient to create dynamic structures that represent both objects that can be derived from the relations in the database and objects that are dynamically generated. In the case of objects that are statically represented by facts

Table 7: Translation

Id	Translation
thrA	MRVLKFGGTSLANAERFLRVADILESNAQQG ...
thrB	MVKVYAPASSANMSGFDVLGAAVTPVDGALL ...
thrC	MKLYNLKDHNEQVSFAQAVTQGLGKNQGLFFP ...

in the database, it is necessary to provide rules for generating the corresponding terms. In this section, we discuss how such rules are provided and how one can construct an interface to the attributes of objects. If one is not interested in these details, he may skip directly to the Summary.

One of the interesting features of logic programming is that a datum (term) and a program (clause) are configured in a uniform structure. For example, a term `:-'(A, B)` can be read as clause **A :- B**.. This feature allows an easy exchange between the data domain and the program domain. The following clauses illustrate how to exploit this uniformity to create terms to represent objects:

```
object(clone(Id-Genome),
        clone(Id, Genome, Chromosome, Pos, Length)) :-
        clone(Id, Genome, Chromosome, Pos, Length).

object(rsite(Genome),
        rsite(Genome, Location, Enzyme)) :-
        rsite(Genome, Location, Enzyme).

object(mapped_gene(Id-Genome),
        mapped_gene(Id, Genome, Chromosome, Location, Direction)) :-
        mapped_gene(Id, Genome, Chromosome, Location, Direction).

object(exon(Id-Genome),
        exon(Id, Genome, Chromosome, No, SeqFragment, Begin, End)) :-
        exon(Genome, Chromosome, Id, No, SeqFragment, Begin, End).

object(structural_gene(Id-Genome),
        structural_gene(Id, Genome, Chromosome, Location, Direction)) :-
        structural_gene(Id, Genome, Chromosome, Location, Direction).

object(translation(Id-Genome),
        translation(Id, Genome, Chromosome, Translation)) :-
        translation(Genome, Chromosome, Id, Translation).
```

Once routines have been created to process generic objects, it becomes fairly easy to extend the system by allowing the introduction of new types of objects—for example, objects to represent regions of sequence, patterns, or restriction fragments. In each case, one simply creates a term to represent the object. In the next section, we consider the issue of how to provide a uniform access to the attributes contained in objects of varying structure.

6.2 Providing an “Object-oriented” Interface

A number of different types of “objects” have been introduced. As more types are introduced, it may become difficult for programmers to remember the exact form of each type of object. Further, one would like to create general routines that can determine whether two objects are known to overlap, to print objects, to scan for patterns in objects, etc. To allow one to construct uniform, general routines to process objects, one should create a general interface to the attributes of objects.

First of all, let us define an index table of attributes for each object shown in Table 1. The predicate `get_attribute/3` relates, for each kind of object, an individual attribute name and its value; here are the rules to implement this access function for objects representing clones:

```
get_attribute(clone(Genome, Chromosome, Id, Location, Length),
               genome, Genome).
get_attribute(clone(Genome, Chromosome, Id, Location, Length),
               chromosome, Chromosome).
get_attribute(clone(Genome, Chromosome, Id, Location, Length),
               id, Id).
```

```
get_attribute(clone(Genome, Chromosome, Id, Location, Length),
              length, Length).
```

Similarly, other objects may be provided with their index tables. It is useful for most objects to have some limited set of common attributes, like "length" and "location." In some cases, this value is not included explicitly in the term representing the object, but can be computed from the term. For example, one specifies a rule for restriction sites as follows:

```
get_attribute(rsite(Genome, On, Offset, Enzyme), length, Length) :-
    length_of_restriction_site(Enzyme, Length).
```

where `length_of_restriction_site/2` is a distinct Prolog rule or fact that returns the length of a restriction site, given the restriction enzyme.

7 Summary

The process of defining and implementing a genomic repository is both difficult and time consuming. However, a straightforward approach based on logic programming does exist that substantially reduces the required effort. In this report, we have outlined the required steps, offered a brief introduction to Prolog, and presented enough guidance to allow a biologist to begin building an integrated representation of maps and sequence data for multiple genomes. Precisely such an integration will be required to support the genomic analysis necessary to extract meaning from the growing body of data.

References

- [1] Bachmann, B. J. (1990), "Linkage Map of *Escherichia coli* K-12, Edition 8," Microbiological Reviews, Vol. 54, No. 2, 130-197.
- [2] Baehr, A., G. Dunham, A. Ginsburg, R. Hagstrom, D. Joerg, T. Kazic, H. Matsuda, G. Michaels, R. Overbeek, K. Rudd, C. Smith, R. Taylor, K. Yoshida, and D. Zawada (1991), "An Integrated Database to Support Research on *Escherichia coli*," Technical Report ANL-92/1, Argonne National Laboratory, Argonne, Illinois, January 1992.
- [3] Baehr, A., R. Hagstrom, D. Joerg, R. Overbeek (1991), "Querying Genomic Databases," Technical Report ANL/MCS-TM-155, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, September 1991.
- [4] Bratko, I. (1986), "Prolog, Programming for Artificial Intelligence," Addison-Wesley Publishing Company, Wokingham, England.
- [5] Kohara, Y., K. Akiyama, and K. Isono (1987), "Application of a New Strategy for Rapid Analysis and Sorting of a Large Genomic Library," Cell 50, 494-508.
- [6] Kröger, M., R. Wahol, and P. Rice (1991), "Compilation of DNA sequences of *Escherichia coli* (update 1991)," Nucleic Acids Research 19: 2023-2043.
- [7] Munz, P., K. Wolf, J. Kohli, and U. Leupold (1989), "Genetic Overview," in: *Molecular Biology of the Fission Yeast*, ed. A. Nasim, P. Young, and B. F. Johnson, Academic Press, New York.
- [8] Rudd, K., W. Miller, J. Ostell, and D. A. Benson (1990), "Alignment of *Escherichia coli* K12 DNA sequences to a genomic restriction map," Nucleic Acids Research 18: 313-321.
- [9] Sanderson, K. E., and J. A. Hurley (1987), "Linkage Map of *Salmonella typhimurium*," *Escherichia coli* and *Salmonella typhimurium*, ed. Neidhardt, F. C., Vol. 2, American Society for Microbiology, Washington, D.C.

- [10] Wickner, R. B. (1981), "Killer Systems in *Saccharomyces cerevisiae*," in: *The Molecular Biology of the Yeast Saccharomyces: Life Cycle and Inheritance*, ed. J. N. Strathern, E. W. Jones, and J. R. Broach, Cold Spring Harbor Laboratory, Cold Spring Harbor.
- [11] Yoshida, K., and C. Smith (1991), unpublished information.

END

**DATE
FILMED
4 12 81 92**