Report 83-12
Stanford -- KSL

Scientific DataLink

Knowledge-Based Simulation of Genetic Regulation in Bacteriophage Lambda.
Scott Meyers, Peter Friedland,
Aug 1983

card 1 of 1

Volume 12 Number 1 1984 Nucleic Acids Research

Knowledge-based simulation of genetic regulation in bacteriophage lambda

Scott Meyers[1] and Peter Friedland

MOLGEN Project, Department of Computer Science, Stanford University, Stanford, CA 94305, USA

Received 12 August 1983

ABSTRACT

We have developed a general-purpose computer program for the functional simulation of regulatory genetics. This simulator is knowledge-based and was developed using the Unit System, a software tool for the acquisition, representation, and manipulation of hierarchically organized knowledge. The advantages of a knowledge-based design are presented, and the simulator's architecture is described. Its performance on the decision between lytic and lysogenic growth in Bacteriophage Lambda is reported.

INTRODUCTION

The development of flexible, easy-to-use simulation programs can provide the experimental biologist with powerful tools for the elucidation of the functioning of complex natural systems. Simulators serve two major purposes: the first is the verification of scientific theories; the second is experimental result prediction. The verification function is called upon when existing theories are being extended or new theories are being generated to explain experimental data; the predictive capabilities are used to predict laboratory results in order to eliminate a great deal of experimental effort.

An especially important role for a simulation program would be as part of a larger system employing artificial intelligence techniques to develop models of a biological system based on experimental observations. Such a program would accept as input observations of a system and would produce as output a model for the system that could account for the observations. The simulation portion of such a program would be a crucial tool for ensuring that only theories that were consistent with the data were developed. It is a major research goal of the MOLGEN project to explore methods for building a system to perform this kind of automatic theory formation.

Traditional Approaches

The most widespread approach to genetic simulation is based on developing large systems of differential equations that predict protein and nucleic acid levels as a function of time. An example of this approach applied to Bacteriophage Lambda can be found in [1]. This methodology results in quantitative predictions, but suffers from the drawback that no functional information is provided. It is

Reprinted with permission from Nucleic Acids Research, Vol.12, No.1, pp.1-9, 1984.

also restricted to systems where a great number of kinetic parameters are accurately known, and such is frequently not the case.

Another approach to genetic simulation is based on modeling "genetic control circuits" as sets of logic equations and predicting sequences of genetic events by searching for stable states in a matrix derived from the equations [2]. While this approach results in functional predictions, much of the information inherent in the system is lost in the abstraction of representing biochemical entities as boolean variables. As a result, it is difficult to answer *why* a system behaves in a certain way without describing how the logic equations were constructed.

Knowledge-Based Approach

Our work has applied a *knowledge-based* approach to the problem of biological simulation. The distinction between a system founded on a database and one founded on a knowledge-base is somewhat fuzzy, but the following list should serve to point out some of the basic differences.

- **Content**: knowledge bases contain both factual and procedural information, while databases tend to contain only factual information. Futhermore, the factual information in knowledge bases varies from simple symbols (numbers, strings, and the like) to much more complicated forms (DNA restriction maps, electron density maps, etc.). An important type of information that is common in knowledge bases but rare in databases is *meta-knowledge*: knowledge about how to use other knowledge. An example of such meta-knowledge is heuristics for determining how relevant and trustworthy other knowledge is under a variety of circumstances.

- **Structure**: databases most often have a rigid, well-defined format that can be expanded or compressed, but cannot fundamentally be changed. Most references in databases are explicit. Knowledge bases, in contrast, have a very fluid, dynamic structure, and many references are contained implicitly.

- **Complexity**: the information in databases often lends itself to rather straightforward analysis, while the information in knowledge bases can be used as a basis for "reasoning" about poorly defined problems. For example, databases have been used to find statistical correlations between symptoms and diseases [3], while knowledge bases have been used in the development of programs to perform automatic diagnosis of human illnesses [4, 5].

Our work has shown that a knowledge-based approach can have significant advantages over the alternative methodologies mentioned above. Unlike a simulator based on differential equations, our simulator is not dependent on detailed kinetic data, and unlike a simulator based on logic equations, our simulator retains all information inherent in the system. A knowledge-based system can take advantage of *all* data about a system, both declarative and procedural. This allows our simulator to predict not only *what* happens, but to predict *why* it happens, as well.

THE SIMULATOR

The simulator was developed using the Unit System [6, 7], a system for the acquisition, representation, and manipulation of hierarchically organized knowledge. It was chosen for use on this project because it has a well-developed user interface that is customized for the realm of molecular genetics, and because it contains a *Rules Language*, an English-like language that allows specialists unfamiliar with computers to describe procedural knowledge directly to the knowledge base.

The simulator knowledge base contains two kinds of information. The first is general information about molecular genetics (e.g., the concepts of *genes, proteins, transcription*, etc.), the second is specific information about the particular biological system under study, in this case Bacteriophage Lambda.

The general knowledge consists of rules in the Rules Language; it is the program that makes the simulator run. These rules are used to determine whether genes are being transcribed, whether proteins are present, etc. Two aspects of these rules are noteworthy. The first is that they are completely general to regulatory genetics - they are not limited to a simulation of Bacteriophage Lambda. It is possible for the simulator to determine the state of any set of genes and proteins that are described in the system-specific portion of the knowledge base. The second is that the rules are designed for geneticists, rather than for computer programmers: it is relatively easy for a scientist familiar with molecular genetics and the general design of the knowledge base to determine what each statement in the Rules Language accomplishes. The user does not need to understand any of the internal details of knowledge representation and manipulation within the Unit System to "program" using the Rules Language. In addition, the language postpones the determination of *context* until the time the rules are actually executed. This means that the user can describe procedural information about objects and properties that do not yet exist; the rule interpreter heuristically "binds" all variables to plausible items in the relevant units being considered. Users can freely employ words like "the," "to," etc. to make statements more readable.

For example, the rules used to determine the current status of a protein are shown below. Comments are in italics.

```
IF BEING-TRANSCRIBED@GENE IS FALSE THEN SET NEWLIFE TO -1 ELSE SET NEWLIFE TO LIFESPAN
```

This rule determines a tentative value for the change in time that the protein of interest will remain active and stores that value in the variable called NEWLIFE. This value is analogous to the protein's change in concentration, for the simulator equates effective concentration with a positive remaining-time. If the protein is not being produced (if the corresponding gene is not being transcribed), the protein's remaining-time (concentration) decays by one unit. Otherwise, the protein's remaining-time is set to the maximum possible, its lifespan.
The "@" notation is provided to allow the user to explicitly specify the context of a variable; in this case BEING-TRANSCRIBED relates to GENEs.

```
SET MODULATOR-EFFICIENCY TO 0

FOR EACH ROW IN THE MODULATORS TABLE IF REMAINING-TIME@MODULATING-PROTEIN >= 1
THEN SET MODULATOR-EFFICIENCY TO ACTION-COEFFICIENT OTHERWISE SET MODULATOR-EFFICIENCY TO
ACTION-COEFF TIMES REMAINING-TIME@MODULATING-PROTEIN
```

```
Some proteins are affected by other proteins. These modulating proteins are kept in a table for any
protein affected. This rule (and the one prior to it) look at the relative strengths and concentrations
of each of the modulators and use them to determine an overall value for the variable MODULATOR-EFFICIENCY,
a measure of how the modulating proteins affect the remaining-time of the protein in question.

SET NEWLIFE TO NEWLIFE + MODULATOR-EFFICIENCY

SET REMAINING-TIME TO REMAINING-TIME + NEWLIFE
The two preceding statements use the values of NEWLIFE and MODULATOR-EFFICIENCY to determine
the amount of time that the protein in question will remain active.

IF REMAINING-TIME > LIFESPAN THEN SET REMAINING-TIME TO LIFESPAN
This statement ensures that no protein ever exceeds its maximum possible concentration.

IF REMAINING-TIME > 0 THEN SET PRESENT TO TRUE ELSE SET REMAINING-TIME
TO 0 AND SET PRESENT TO FALSE
A protein is present if it will still be active for some time greater than zero.
```

The information in the knowledge base that is specific to the biological system under study is divided into three classes: genes, proteins, and special DNA control loci (promoters, operators, terminators, and nut sites). Each of these classes is defined as a *prototypical unit* within the knowledge base with *slots* to contain the properties used by the simulator. These slots may either be static facts associated with a particular object, *e.g.*, the *lifespan* (akin to a half-life) or *binding affinity* of a protein, or dynamic properties modified during the simulation, such as whether a gene is currently being transcribed or which, if any, protein is attached to a special DNA locus. The process of describing a genetic system to be simulated consists of creating the appropriate *children* of these units and filling in all of the slots for which information is known.

An example of one of the prototypical units (GENES) is shown below. Each slot occupies one line in the listing, and for each slot, three pieces of information are given: its name, its datatype, and its value. Datatypes describe what kind of information is stored in the slot, for example, a description (for the DESCR slot), a string (for the ORGANISM slot), or a list of other units (for the PROMOTERS slot). For a prototypical unit, slot values may be either an actual *instance* of the datatype (for the CREATOR slot), or a *restriction* on possible values for an instance of the datatype (a choice of "LAMBDA" or "E-COLI" for the unit's ORGANISM, for example). As in the prior listing, comments are italicized.

```
Name              Datatype        Value

DESCR:             <DESCR>      This node is the root of all genes.
-----
CREATOR:           <CREATOR>    "CSD.MEYERS"
The above two slots (as well as several others not shown) contain "bookkeeping"
information associated with the unit.
-----
ORGANISM:          <STRING>     One of: ["LAMBDA" "E-COLI"]
-----
PROMOTERS:         <LIST>       UNITs:
There may be several promoters associated with a gene. This slot will be filled
for a specific gene with a list of the names of one or more specific promoter units.
-----
INTERVENING-TERMINATOR:
                   <UNIT>
If there is a terminator locus between a gene's promoter and its coding region,
the name of that locus will appear here.
```

```
-----
NUT-SITE:          <UNIT>
If a nut site occurs within the coding region of a gene, the name of that site
will be entered here.
-----
MUTATED:           <STRING>    One of: ["TRUE" "FALSE"]
-----
BEING-TRANSCRIBED:
                   <STRING>    One of: ["TRUE" "FALSE"]
The above two properties are used by the simulator to indicate the current state
of the gene.
```

The overall structure of the knowledge base used to simulate Bacteriophage Lambda is shown in Figure 1. It is important to note that both the rules for the simulator (the "program") and the data about Lambda (*i.e.*, the genes, proteins, and DNA loci) are stored in the same knowledge base. This type of uniformity in representation would be virtually impossible if a database were used.

APPLICATION TO BACTERIOPHAGE LAMBDA

Bacteriophage Lambda is an attractive genetic system for simulation studies for two reasons. First, it exhibits a variety of regulatory mechanisms that are believed to be common to all biological systems. Lambda is thus the subject of intense interest in the biological community. Second, Lambda has been studied for over two decades and is now probably better understood than is any comparable organism. This paper presupposes the reader's familiarity with the basic interactions that take place in the regulatory region of Lambda during the process that leads to lysis or lysogeny; those unfamiliar with the system may want to consult [8].

The Model of Lambda

Only those genes, proteins, and DNA control loci which affect the decision to go lytic or lysogenic were included in the model. This led to the simplified model of 6 Lambda genes, 7 promoters, 3 operators, 3 terminators, and 2 nut sites shown in Figure 2. In addition, the *hfl* gene of *E. Coli* was included in the model, for, unlike most *E. Coli* genes, the interaction of the *hfl* gene product with the Lambda proteins is both important and well understood.

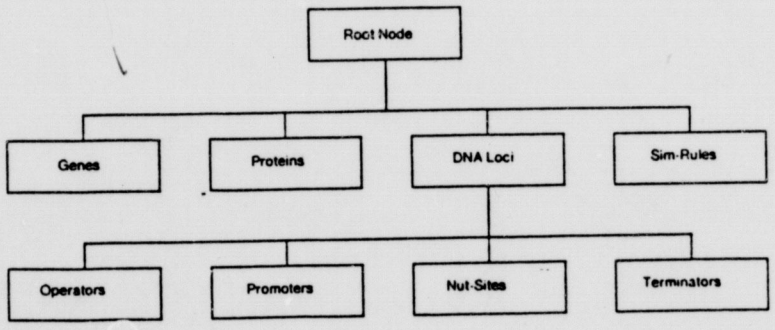

Figure 1: Structure of the Lambda Knowledge Base

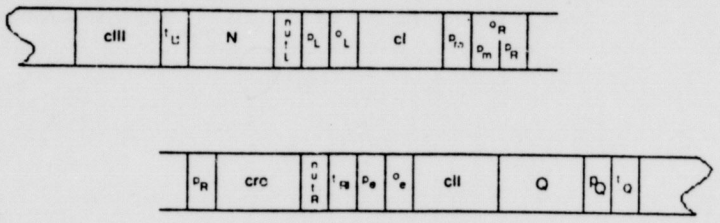

Figure 2: Regulatory Region of Simplified Lambda Genome

Results

We ran simulations on eight different genetic systems: wild-type Lambda; single-locus mutants of Lambda at *cI*, *cII*, *cro*, *N*, and nut_R; a Lambda double mutant at *N* and t_{R1}; and wild-type Lambda in an *hfl*⁻ host. All mutations were assumed to be completely deleterious (the affected DNA locus suffered complete loss of function).

In all but two of these cases, the results of the simulator agreed with what is observed in the laboratory. Those two examples, the *N* mutant and the nut_R mutant, resulted in a prediction that the system would oscillate between two different genetic states without making a committment to either the lysogenic or the lytic pathway. Within the confines of the model, this is a correct prediction, for the current model assumes entirely deterministic behavior. In reality, of course, this is not the case. In particular, terminator sites do not stop all transcripts that have not been explicitly antiterminated; they are "leaky," a fact not taken into account by our model. Adding nondeterministic behavior to the model would significantly increase its power; we plan to make this enhancement in the future.

The knowledge-based simulator of Lambda also provides a useful tool for teaching students about regulatory models. The student can describe his current understanding of Lambda to the simulator, run the system under various conditions, and see if the results reflect reality. Where conflicts occur, the student can determine exactly what knowledge was incorrect or incompletely understood, and modify the knowledge base accordingly.

The use of the simulator for educational purposes can be extended to encompass serious scientific study of the model by experts. Presumably, if an expert's view of the Lambda system does not reflect reality when the simulator is run, then the model itself must be improved. As was mentioned earlier in the article, emulating the human process of scientific model construction, testing, and improvement is a major research goal of the MOLGEN project; the knowledge-based simulator will be an important tool for this research.

Significantly, the Lambda knowledge base (including the simulator rules) took on the order of only two man-months to build and test. The existence of a sophisticated knowledge representation and acquisition tool (the Unit System) allowed the construction of a simulator in far less time than would have been required with conventional programming techniques.

AN EXAMPLE

The following example illustrates the type of information that the simulator provides to the biologist. The ability to describe the functional state of each important entity (e.g. promoter, protein, etc.) at all times is one of the major advantages of the knowledge-based approach. The example shows the results of running the simulator on the *N* and t_{R1} double mutant. Annotations to the transcript are printed in italics.

```
ENTER LIST OF MUTATED UNITS:
  n t-r1
```

The user is first asked to provide a list of mutations.
The mutations are specified by listing the names of the units
in the knowledge base that represent the DNA loci to be mutated.
Entering the null list (no entries) results in a simulation of
wild-type Lambda.

```
ENTER NUMBER OF PHAGE TO SIMULATE:
  1
```

The only other information specified by the user is the number
of bacteriophages to simulate. Because the model is deterministic,
all simulations for any given set of mutations will result in identical
behavior. When nondeterminism is added to the model, however, it will
be possible to obtain a distribution of results.

```
-------------------------------------------------------
** STARTING ITERATION NUMBER 1 **
```

The user is informed every time the simulator begins a new cycle.
A cycle consists of updating the state of the system and then checking for
a commitment to lysis or lysogeny.

```
** P-E IS INACTIVE **
** P-L IS ACTIVE **
** P-M IS INACTIVE **
** P-R IS ACTIVE **
```

The state of each of the promoters is printed out after they have
been updated. No mention is made of the operators, for if a promoter
is not active, genes originating at that point cannot be transcribed.

```
** CI IS NOT BEING TRANSCRIBED **
** CII IS BEING TRANSCRIBED **
** CIII IS NOT BEING TRANSCRIBED **
** CRO IS BEING TRANSCRIBED **
** HFL IS BEING TRANSCRIBED **
** N IS NOT BEING TRANSCRIBED **
** Q IS BEING TRANSCRIBED **
```

After the status of the genes is updated, their states are also printed out.

```
** REMAINING TIME OF CI-PROTEIN IS 0 **
** REMAINING TIME OF CII-PROTEIN IS 3.0 **
** REMAINING TIME OF CIII-PROTEIN IS 0 **
** REMAINING TIME OF CRO-PROTEIN IS 3.0 **
** REMAINING TIME OF HFL-PROTEIN IS 3.0 **
** REMAINING TIME OF N-PROTEIN IS 0 **
** REMAINING TIME OF Q-PROTEIN IS 4.0 **
```

The final piece of information for an iteration is the remaining time of each
of the proteins. The remaining time represents the number of cycles that a protein's
concentration will remain sufficient for it to exert an influence on the system.
The remaining times shown for cII, cro, hfl, and Q are the same as their lifespan,
since no time has yet passed since their production.

```
-------------------------------------------------------
```

```
** STARTING ITERATION NUMBER 2 **
** P-E IS INACTIVE **
** P-L IS INACTIVE **
** P-M IS INACTIVE **
** P-R IS INACTIVE **
** CI IS NOT BEING TRANSCRIBED **
** CII IS NOT BEING TRANSCRIBED **
** CIII IS NOT BEING TRANSCRIBED **
** CRO IS NOT BEING TRANSCRIBED **
** HFL IS BEING TRANSCRIBED **
** N IS NOT BEING TRANSCRIBED **
** Q IS NOT BEING TRANSCRIBED **
** REMAINING TIME OF CI-PROTEIN IS 0 **
** REMAINING TIME OF CII-PROTEIN IS 1.6 **
** REMAINING TIME OF CIII-PROTEIN IS 0 **
** REMAINING TIME OF CRO-PROTEIN IS 2.0 **
** REMAINING TIME OF HFL-PROTEIN IS 3.0 **
** REMAINING TIME OF N-PROTEIN IS 0 **
** REMAINING TIME OF Q-PROTEIN IS 3.0 **
```

Note how the remaining times have changed. Hfl, being an E. Coli gene, is always transcribed, and the remaining time of its protein is therefore always the same as its lifespan. The proteins from cro and Q have begun to decay and their remaining time has decreased by one time unit over the course of the second iteration. Because the hfl gene product attacks the cII protein, the decay rate of the latter gene product is faster than normal. Its remaining time has thus decreased by 1.4 time units in the past cycle.

Iterations 3 and 4 are omitted here for brevity.

```
** STARTING ITERATION NUMBER 5 **
** FINISHED SIMULATION NUMBER 1 **
```

Having come to the conclusion that the phage being simulated would pursue lytic growth, the simulator announces that it has finished the simulation.

```
*** SIMULATION RESULTS ***
NUMBER OF LYSOGENIC PHAGE =  0
NUMBER OF LYTIC PHAGE =  1
PERCENTAGE LYSOGENIC PHAGE =  0.0
PERCENTAGE LYTIC-PHAGE =  100.0
```

When all simulations are completed, the simulator prints a summary of the results.

ACKNOWLEDGEMENTS

This work is part of the MOLGEN project, a joint research effort among the departments of Computer Science, Medicine, and Biochemistry at Stanford University. The research has been supported under NSF grant MCS80-16247. Computational resources have been provided by the SUMEX-AIM National Biomedical Research Resource, NIH grant RR-00785-08, and by the Department of Computer Science.

Thanks are due to Dr. René Bach for valuable comments during the development of the simulator and during the preparation of this manuscript.

MOLGEN is a trademark of the Board of Trustees of Stanford University.

[1]Current address: Silvar-Lisco, 3172 Porter Drive, Palo Alto, CA 94304, USA

REFERENCES

1. Kananyan, G. K., *et.al.*, "Expanded Model of the Ontogenesis of Phage Lambda," *Soviet Genetics*, Vol. 16, 1980, pp. 1280-1286.
2. Thomas, R., *et. al.*, "A Complex Control Circuit: Regulation of Immunity in Temperate Bacteriophages." *Eur. J. Biochem.*, Vol. 71, 1976, pp. 211-227.
3. Blum, R.L., "Discovery, Confirmation, and Incorporation of Causal Relationships from a Large Time-Oriented Clinical Database: the RX Project," in *Computers in Biomedical Research*, H.R. Warner, ed., Academic Press, New York, 1982, pp. 164-187.
4. Shortliffe, E.H., *Computer-Based Medical Consultations: MYCIN*, Elsevier/North Holland, New York, 1976.
5. Miller, R.A., *et. al.*, "INTERNIST-1, An Experimental Computer-Based Diagnostic Consultant for GEneral Internal Medicine," *New England Journal of Medicine*, Vol. 307, 1982, pp. 468-476.
6. Smith, R.G. and Friedland, P.E., "Unit Package User's Guide," Heuristic Programming Project Memo HPP-80-28, Stanford University, 1980.
7. Stefik, M., "An Examination of a Frame-Structured Representation System," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, IJCAI, 1979, pp. 845-852.
8. Hershey, A. D., ed., *The Bacteriophage Lambda*, Cold Spring Harbor Laboratory, Cold Spring Harbor, New York, 1971.