Research Proposal Submitted to the National Science Foundation

Proposed Amount $294,476 Proposed Effective Date 6/1/78 Proposed Duration (months) 24

Title MOLGEN: A Computer Science Application to Molecular Genetics

Principal Investigator Edward A. Feigenbaum
Soc. Sec. No. [redacted]

Submitting Institution Stanford University

Department Computer Science

Institution (if different from submitting institution)

Branch/Campus

Address Stanford, California 94305

Branch/Campus

Co-Principal Investigator Joshua Lederberg Soc. Sec. No. [redacted]

Co-Principal Investigator Soc. Sec. No.

If renewal request previous NSF grant No. MCS 76-11649

Make grant to Stanford University
(name of institution or organization to which grant should be made)

Endorsements:

Principal Investigator(s)	Dept. Head	Institutional Admin. Offic
Name Edward A. Feigenbaum		
Signature Edward A. Feigenba	Edward A. Feigenba	
Title Professor and Chairman		
Telephone Number 415-497-4079		
Date October 27, 1977		

October 27, 1977

Table of Contents

1 Introduction

This application addresses the continuation of research on the applications of artificial intelligence (AI) (1) to experimental molecular genetics. It is an extension of a longstanding effort to cultivate attention to ongoing laboratory research as a domain of explorations in artificial intelligence. Our major effort in this field had been in the DENDRAL project, with analytical organic chemistry as the object discipline. The present effort, "MOLGEN", focuses on a new object domain, namely molecular genetics. However, for reasons that will be elaborated, the focus is on programs to suggest experiment-planning sequences needed to solve a given structure, rather than on hypotheses about the structures themselves, which characterizes DENDRAL.

Our primary motivation is to deepen our knowledge of the art and science of creating programs that reason with symbolic knowledge to aid human problem solvers. The task domain--molecular genetics--serves as a rich intellectual and scientific environment in which to develop and test our ideas.

The major computer science issues we are addressing are:

Present Grant period (June 1976 - June 1978)

(1) Creation of a knowledge representation system with a knowledge acquisition package. The system, known as the Units Package, may be used to build a knowledge base in any suitable domain. It provides an object-centered approach for storage of both declarative and procedural information concerning all entities in the domain. Section 1.2.1

(2) Structured representation of process information. Procedures which simulate the action of the various processes in the domain form an integral part of the knowledge base. Moreover, the representation framework allows for inspection and acquisition of those procedures. Section 1.3.3.1.

(3) Creation of program schemata and instances for general problem solving steps. Domain-independent knowledge about general problem solving methods also fits into the knowledge representation structure we have devised. Section 1.3.3.3.

---

(1) For definitions of many of the genetic and computer science terms used throughout this proposal, see Appendix I.

(4) Domain Specific Critics. Mechanisms for the activation of various domain specific strategies when certain predefined situations occur during the course of experiment design. Section 1.3.3.4.

(5) Development of a specific planning strategy designed to provide high-performance for the class of genetic experiments known as discrimination experiments. The idea is based on indexing abstracted experimental designs to the types of structural features for which they have proven useful. Section 1.3.4.

Grant Renewal Period (June 1978 - June 1980)

(6) Creating a more comprehensive genetics knowledge base. Expanding the knowledge base within the area of DNA structural manipulation problems. This work will be done mainly by the Stanford Genetics Department. Section 2.1.

(7) Abstracting and Saving Plans. Recognizing when newly-created experiment designs are worth saving and then generalizing those plans so they are useful for more than the specific problem environment which caused their generation. This work will be done in the Computer Science Departments of both Stanford and UNM. Section 2.2.

(8) Making use of the process of hypothesis formation to help debug MOLGEN-produced experiment designs. This process is especially important in a domain like molecular genetics where incomplete knowledge about objects and processes is the rule rather than the exception. This work will be done in the Stanford Computer Science Department. Section 2.3.

(9) Experiment planning by analogy. MOLGEN provides an excellent environment for exploring various types of analogical reasoning. We integrate problem-solving by analogy into the experiment design system as one of the possible tools for solving subproblems. This work will be done in the UNM Computer Science Department. Section 2.4

(10) Performance evaluation as an integral part of the knowledge representation and acquisition system. We view the process of evaluating a system's performance and suggesting improvements as an AI problem solving task. The strategies for this evaluation will be

stored within the same framework as all other MOLGEN planning strategies. This work will done in both the Stanford and UNM Computer Science Departments. Section 2.5.

We have designed our effort to facilitate generalization to other domains beyond genetics in future research and applications.

A second motivation behind the proposed research is to develop tools that can benefit molecular geneticists. We believe there is substantial benefit to be derived from programs that act as "intelligent assistants" to scientists. First of all, the sheer amount of detailed knowledge a scientist is expected to know makes it likely that good experiments are being missed. Second, we believe that an intelligent planning assistant can offer some help in reasoning about the consequences of combining experimental facts in many possible ways.

A third motivation for applying artificial intelligence techniques to an experimental science like molecular genetics is to help us better understand the scientific method. The rigorous detail required for creating computer programs that assist in the performance of scientific tasks forces us to explicate concepts and procedures much more carefully than practicing scientists usually do.

During the current grant period, the basic MOLGEN system will be completed. (See Section 1.2 for details). This system will have a modest genetics and problem solving knowledge base, an ability to design a subclass of genetics experiments and a representation and acquisition system capable of handling all of the knowledge used by an experiment design system. During the renewal period we will expand the system's expertise and use the system to explore new problem solving capabilities. (Readers can skip to Section 2 for details of work to be completed during the renewal period). The basic MOLGEN system will provide an excellent framework for exploring more sophisticated experiment design activities and problem solving techniques. We have found in our analysis of the activity of geneticists that designing experiments is intimately coupled with forming hypotheses about the failure of the experiment design in the laboratory, then testing the hypotheses and, as a result of the tests, gaining new knowledge wtich effects the design. The new hypothesis formation component of MOLGEN will extend problem solving capabilities in this direction. Another important element in the repertoire of an expert human problem solver is the ability to form analogies between problems and to use the analogy to solve new problems. Our exploration of analogical reasoning within the MOLGEN system will lead to a new problem solving component integrated with the rest of the design system. New expertise will come in a direct manner from the expansion of the knowledge base by geneticists. At the same time we will increase our problem solving knowledge base by adding a component which monitors the problem solving

experience of the system and abstracts new experiment designs for the design process. The generalization process developed for this monitor will be used by the analogical reasoning component. Our final extension to the MOLGEN system is the ability to evaluate its own performance. This evaluation will aid the development of all of the other extensions.

1.1 Motivations

The successful initiation of and perseverance with interdisciplinary research of this kind can be hindered or aided by many intangible factors in the local context, as well as by the personal idiosyncrasies of the participants. We need only point to the history of the DENDRAL project as evidence of our proven capability, for whatever reasons, to engage in such research in the Stanford University environment. It is inevitably true that no one person can embrace a professional level of critical insight over all the fields represented, but this has been surmounted by dint of hard work on mutual communication from all sides. More problematical is the problem of communicating our findings to a disparate group of disciplinary experts outside, e.g. to computer scientists and to molecular geneticists who customarily have had little need of perusing each other's disciplines. The best solution is to get these parties engaged not just in reading proposals or publications, but in the actual manipulation of the working programs -- particularly with the help of computer networks that link dispersed sites. Even so, an amount of technical detail that could well try the patience and skill of the most tolerant reviewer is unavoidable if we are to offer a concrete, fair picture of the status of our efforts, especially in the formative stages. We trust that, as is often true, the maturation of these efforts can be accompanied by a simplification in presentation, and meanwhile we must appeal to the patience and good will of constructive critics. The supporting documents attached to this proposal offer a sample of our working papers and publications. Further documentation on any aspect of the present system is, of course, available.

1.2 Progress To Date

The following sections describe our progress towards developing an experiment planning program. The first part discusses a knowledge management system. Most of our design and programming effort has been in this area. A package of programs for the representation and acquisition of knowledge is discussed and our progress in using it in creating a genetics knowledge base is also described.

The next section focuses on our examination of genetics knowledge. It discusses various studies we have made of genetics experiments and what we have learned from them. A short section describes some work we did in representing the action of the ligase enzyme and how this was of benefit to work in Prof. Lederberg's laboratory.

1.2.1 Knowledge Base Research

The success of MOLGEN as an experiment planner will depend on the quality of its knowledge base. Therefore, much of the research effort to date has been in the design and implementation of a knowledge representation and acquisition system. All of the information relevant to the planning process will be an explicit part of the knowledge base. The motivation for this aspect of the design is the necessity to expand the program capabilities in a modular fashion and to explain the rationale behind the program's planning behavior. We need to represent concepts (e.g. enzyme), instances (e.g. EcoRI), relationships among concepts, and relationships among instances. In addition, we need to represent processes. Two recent papers [42][3] have delineated problems which arise in interpreting a knowledge base when the semantics of the representation are not clearly specified. We have purposely limited the expressive power of our representations to enable us to clearly define their semantics.

The result of this work is the Unit Package. Although this package has been designed in the context of our genetics application, the package does not contain any genetics knowledge. At this time one other group within the Stanford Heuristic Programming Project, the AGE group, is using the package.

Our knowledge base design has been reported in the literature [24]. Other documents relating to its use are included in the appendices. Also included is a sample session illustrating the acquisition of a DNA structure and a current summary of part of the knowledge base dealing with enzymes. Section 1.2.1.1, Section 1.2.1.2, and Section 1.2.1.3 describe the implemented portions of the representation package. Section 1.3.1 discusses some of the further developments for the knowledge base that are planned for the rest of the current grant period.

1.2.1.1 The Unit Package: Background and Relation to Other Research

Design of the representation package (termed the Unit Package) was started in November 1976 although most of the programming has been done since March 1977. The design of the Unit Package profited from

the experience of other efforts e.g. KRL-0 [1], SMALLTALK [20], and the work at the MIT AI Lab on frames [27][21]. One important difference in perspective between the Unit Package research and many other efforts is that it is at once a knowledge acquisition system and a knowledge representation system. The knowledge acquisition system is in the same spirit as Davis' work on TEIRESIAS [7] where schemata were used to guide the acquisition of knowledge. Another distinguishing feature of the Unit Package is that it is also used to represent and acquire process information, that is, action and strategy knowledge (see Section 1.3.3.1). The package performs many other knowledge management tasks for the system.

The basic element of our representation is an entity called a unit. Units are either "prototypes" or "instances". Prototypes are used to represent Woods' intensions [41], Brachman's concepts [3]. That is, a prototype defines a class, that is, the knowledge expected for a particular concept in terms of slots with the appropriate fields (e.g. role, value/specification, default). (2) Individuals are defined as instances of prototypes. Subclasses can be formed with a special link called generalization/specialization. Concepts can only have one generalization and instances only one prototype. Other relationships among units can be expressed by the value/specification of a slot or by creating an explicit relation unit.

The knowledge acquisition system is used to acquire both prototypes and instances. This system is being used by system builders to create the bootstrap network described in the next section and by Jerry Feitelson, our genetics research assistant, for defining and instantiating knowledge about several families of enzymes. (See Appendix II and the supporting documentation for examples of knowledge acquisition.)

Other knowledge base management chores include automatic bookkeeping and knowledge integration. The bookkeeping chores simply keep a record of all pertinent information about the creation or modification of a unit.

1.2.1.2 The Bootstrap Knowledge Base and Object-centered Perspective

One important aspect of the design of the system is that the knowledge base contains knowledge about its own data representations. We have provided what we term a "bootstrap knowledge base." It contains domain independent knowledge about commonly used data types. When

---

(2) The slots in a prototype are similar to Brachman's DATTR links. A limited version of Brachman's MODALITY link is implemented. MODALITY specifies the necessity of the specific DATTR in instances.

using our knowledge base in a new domain, an artificial intelligence researcher would probably start with the bootstrap knowledge base and then proceed to create units for the specific knowledge of his task area. Both the AGE and genetics knowledge bases have been started in this manner.

The bootstrap knowledge base serves to illustrate our approach to extensibility. Most of the bootstrap knowledge base is made up of the procedures which capture the knowledge in the system about primitive datatypes. To add a new datatype to our system, one needs to provide the knowledge base with procedures for some basic operations -- such as editing and printing. Actually, the same approach is used in the unit package for defining a new datatype as is used for defining a new enzyme. The process of defining new datatypes requires, however, an understanding of Interlisp because the primitive processes in the system are grounded in that language. New datatypes must be defined together with their basic operations and entered into the knowledge base.

The approach used for attaching the basic operations to the primitive data types is very similar to SMALLTALK [20] and KRL-0 [1]. As far as the knowledge base is concerned, in order to edit a DNA structure, one sends the DNA unit an edit message. Another way of saying this is that the procedure for editing a DNA structure is attached to the DNA unit. Thus, indexing of the "edit procedure" is via the "DNA object" so that this is an application of the object-centered viewpoint introduced in the languages mentioned above. This organization makes it easy to add new datatypes to the knowledge base.

An object-centered viewpoint is used in our system for other applications as well. An example of this is the "inspector" idea discussed in Section 1.3.3.4 which illustrates the significance of object-centered indexing during planning. In that example, particular knowledge about related planning situations is associated with the genetic object -- "pH". Generally only operations basic to primitive data types are represented in LISP code. Other process information is expressed as units. Instances of process units can also be associated with objects.

1.2.1.3 Considerations of Human Engineering

Because we expect our knowledge base to be maintained by geneticists, it has been important to carefully consider the interaction expected between a geneticist user and our Unit Package. This effort has resulted in the crafting of a package which is particularly easy to use. This ease is verified by the acceptance of our system by both geneticists and users on the AGE project.

One part of human engineering has already been mentioned above. The system automatically records who has entered or modified any of the information in the knowledge base. It provides summaries of what is in the knowledge base and can indicate which areas of the knowledge base have been changed recently. (See the attached supporting documentation for a summary of a recent version of the genetics knowledge base.)

Another example of this human engineering is the "User Profile" maintained in the unit editor. A user's profile contains such information as whether he is a programmer, whether he prefers verbose printouts, whether he likes to be informed when the system does things automatically for him (i.e. that he has not requested). This information helps the system to tailor its interaction with a user. In addition, terminal communication is handled via very general routines which check for type ahead and allow help information to be provided to the user at any point in the dialog. The result of this is that an experienced user can have very brief interactions with very little prompting from the system. An inexperienced user will be prompted at each step of the dialog and may type "?" at any time to get an explanation or example of what the program expects. (See the User's Guide to Units in the attached supporting documentation.)

Many of the human engineering parts of the system take advantage of the facilities of InterLisp. For example, when a programmer is changing some procedures in the knowledge base, he need not explicitly request that the changed functions be recompiled. This part of the updating is performed automatically at the end of a session.

Our final example of human engineering illustrates the fuzzy borderline between the technical problems of making the system easy to use and some more complicated issues of knowledge base management. Recognizing the importance of experimentation in representing knowledge, one of the facilities of the unit system is to merge knowledge bases. This makes it easy to update various knowledge bases when changes are made -- for example when the bootstrap knowledge base is updated. Part of the transfer protocol includes the facility for one unit to imply a requirement for another. For example, suppose that the bootstrap knowledge base has just been updated so that it has knowledge about the representation of an important data structure - say "trees". If this is useful in representing knowledge in the AGE system, they need only request to transfer the "tree unit" from the bootstrap knowledge base. Any other units essential to complete the knowledge about "tree representation or tree operations" will be transferred automatically at the same time.

The more difficult aspect of merging knowledge bases concerns how the system can determine which units should be transferred when a particular unit is transferred. Restated -- how can the system find

all of the relevant knowledge about a unit? Our approach to this involves a method of inserting knowledge about relevance into the knowledge base itself.

Considerations like those in the previous paragraph are simple enough from the implementation point of view and are described in the appendices. They are representative of the effort we have taken to make the units package easy for a geneticist (or a programmer) to use.

1.2.2 Studying the Process of Experiment Design in Molecular Genetics

Considerable effort was given during the first fifteen months of the MOLGEN grant to the study of the process of designing experiments. Attempts were made to determine subsets of the domain which would provide realistic target areas for automated experiment design and to collect the strategies and domain knowledge used by experts for those subsets. Detailed descriptions of the results of this investigation are given in the appendices.

The work included four major efforts:

1. Designing as many rational experimental plans as possible for an experimental problem from Prof. Lederberg's laboratory, the determination of the presence of poly-A sequences in DNA. (See attached supporting documents: "The Embedded Sequence Problem")

2. Collecting "skeletal" or abstracted plans for a wide variety of structural elucidation problems. These plans are meant to capture the re-usable aspects of an experimental plan for use in planning other experiments. They are an essential component of an approach to planning described in Section 1.3.

3. Analyzing in detail the logic and development of a single experiment performed in Prof. Lederberg's laboratory. This study considers both the knowledge which led to the successful experimental design, and also the process of hypothesis formation to account for experimental failures. (See attached supporting documents for a copy of [14])

4. Analyzing a collection of experiments suggested by several geneticists as good examples of logically produced experimental designs in order to extract the knowledge needed for MOLGEN to design the experiments.

Our design of the system has been strongly influenced by these studies. Rather than give detailed results in the body of this proposal, we present a summary of these influences.

1. We will concentrate our initial efforts on experiment planning to the subfield of restriction enzyme experiments and then expand to applications to plasmid and sequencing experiments.

2. We will represent genetic objects so that they may be viewed from a variety of different perspectives. Our studies showed that different perspectives led to different experimental algorithms. (See [40] for a similar finding in program synthesis.)

3. While we are restricting the range of objects and processes as indicated in (1), we must represent many different types of knowledge about each object. Even in a single experiment, a great diversity of knowledge was needed to design the experiment [14].

4. The planning system will be one component of an eventual genetics expert. The experimental design process is much more event-driven than anticipated. The eventual expert system will need a hypothesis formation component and techniques to exploit serendipity. For example, results of a particular transformation may indicate interesting sidelights which could be investigated in the context of the current experiment.

1.2.2.1 Simulating the Process of Enzymatic Ligation

An attempt was made early in the first year of the current grant to simulate the action of enzymatic ligation on DNA structures. There were several motivations behind this effort. We wished to test our newly developed DNA structural representation and to learn something about representing the processes that modify those structures. Also, we were presented with an interesting problem where we could be of actual use to a laboratory geneticist (Dr. S. D. Ehrlich).

The experiment in question was one in which sticky ended DNA of uniform length was reacted with ligase to join together ends. Each ligation could either join two different DNA structures or could circularize a single structure. In the former case the new structure would still have two sticky ends and the total number of structures in the sample (concentration) would be reduced by one. In the latter case

the number of structures would remain the same, but one structure would be "inactivated" for further ligation. The problem was to predict what the total sample population would look like after some portion of the original monomeric structures had disappeared. That is, what percent of the total structures would be monomeric circles, dimers, dimeric circles, and so on. These percentages would be compared with laboratory results in order to determine how many different ends types were present in the sample. (The more different end types, the greater the ratio of circles to linear and shorter structures to longer structures).

The simulation we developed was based on a kinetic theory of ligation [11]. The simulation results correlated remarkably well with the actual laboratory experiment when a model with two different end types was chosen. This gave additional support to Dr. Ehrlich's conclusion that indeed two different structure types had been present. The program was then used for the related experiment of maximizing inserts of a small gene into a plasmid. The idea was to cut the plasmid with the same restriction enzyme that had produced the sticky-ended gene, join the two, and then recircularize the structure to produce a recombinant plasmid. The problem is that many other structures, monomeric circles, long linear concatemers, etc., can occur. The problem was to determine optimum starting concentrations of the two structures and the length of time to carry out the ligating process to maximize the desired product. The suggested values given by the simulation program were of considerable assistance in carrying out the laboratory operation.

The major significance of this work for future MOLGEN development (besides giving a working geneticist assistance and encouraging his cooperation in communicating problem-solving knowledge to us) was that this was our first attempt to capture the knowledge needed to simulate a genetic process. We made no effort to consider environmental side effects like temperature and pH, but the results were still quite encouraging. The major problems we discovered were twofold and related. First, the simulation program was hard to expand to other problems. Adding other relevant information, for example stereochemical effects, or about new processes such as cutting structures, would involve considerable additional programming. Second, all of the knowledge was hidden in the simulation program (written in SAIL). The program could offer no explanation of its predicted results, nor could the user make interactive changes to its guiding theory. These results led to our decision to make all knowledge, not just knowledge about static genetic objects, explicit in our knowledge base. A discussion of how this is being accomplished is given in Section 1.2.1 and Section 1.3.3.

1.3 Plans for the Remainder of this Grant Period

1.3.1 Continuing Knowledge Base Management Research

Our progress in developing a knowledge representation system for MOLGEN has been discussed above. Our continuing effort during the grant period is divided into (1) filling in a knowledge base with entries specific to a specialized subset of molecular genetics, and (2) expanding our representational capabilities to include knowledge about the process of experiment design.

1.3.2 A Genetics Knowledge Base for Restriction Enzyme Experiments

The Units package described above will be used to acquire and store the knowledge needed to plan experiments. The initial knowledge base will be limited to the facts and heuristics necessary to reason about restriction enzyme experiments. This will include:

1. Units for basic concepts like DNA structure, enzymes, and samples.

2. Units for each individual restriction enzyme.

3. Units for laboratory techniques which make use of restriction enzymes.

4. Procedures which describe the action of restriction enzymes on DNA.

After the restriction enzyme knowledge has been debugged, knowledge about the fields of plasmid technology and sequencing DNA will be added.

1.3.3 Knowledge About Processes and Planning

The basic planning work in MOLGEN will be divided into two major efforts. General domain-independent planning strategies will be developed within the units knowledge representation system as part of the process of making an "AI toolbox." At the same time, work will proceed on a specific strategy tailored to provide high-performance for discrimination experiments in molecular genetics. The specific approach for discrimination experiments is discussed below after the discussion of our general approach to process representation and planning.

1.3.3.1 Representation of Processes

Extending the Unit Package to accommodate the representation of processes is the important next step in the development of the knowledge base system. We view both simulation (of a laboratory technique) and planning operations (e.g. selection of a subgoal) as processes. Our approach to the representation of processes has previously been discussed [24][36]; the main ideas are reviewed below.

We want to provide a facility that makes process description and debugging as easy as possible. For example, the system will provide an appropriate set of primitive operations for the operations on DNA models. The geneticists and the computer scientists will create prototypes for laboratory steps. Geneticists will then use these prototypes as guides during the acquisition of knowledge about particular laboratory tools. The same prototype/instance system will be used to represent planning operations.

Many of the processes we want to represent can be grouped into classes such that there is a prototypical action for each class. That is, the individual processes (e.g. action of EcoRI) can be viewed as instantiations of a process concept (e.g. action of restriction enzymes) just as individual objects are the extensions of object concepts. Many researchers in program synthesis (such as [9]) have used a "program schema" to represent a prototypical program. The schema is then instantiated by the synthesis system to produce a concrete program. A program schema consists of a generalized program with abstract predicate, function and constant symbols, input/output specifications, and restrictions on possible instantiations of the abstract symbols.

This notion of a program schema (we will use the expression "process schema" synonymously) can be implemented within the current unit representation. The abstract predicate , function, and constant symbols are slot names. The restrictions on possible instantiations are specified in the value/ specification field. The abstract function symbols may be restricted to be either system primitive functions or instantiations of other program schemata. I/O variables and relationships among them can also be specified via the slot mechanism and the "relation" units. Within a rule unit there is a special slot which contains the generalized procedure. Thus, a program schema can in fact be developed in a modular fashion: first create the simpler, or primitive schemata; then decide on the constant, predicate, and function symbols needed (i.e. the named component parts in a process description); then create the generalized procedure using these symbols as the named components.

Implementation details of rule units are currently being examined. While in synthesis programs, the system creates the

instantiation from program segments, in the current design of MOLGEN, the user creates the instantiation. The acquisition system must be able to check instantiations to make certain that the restrictions and I/O specifications have been met.

The use of process schemata and their representation as units has implications for planning also. The visibility of process components in slots is important so that other processes can examine and select them according to the values in the slots. General problems of such indexing methods and comparisons of systems which use these methods can be found in the literature [8][36].

In many cases it is appropriate to associate the process elements of the system with definitions of the objects. This "object-centered" viewpoint has been expounded [1] and implemented as "attached procedures". Several different applications for attached procedures in MOLGEN are given [24]. These attached procedures can be system primitives or instances of process prototypes.

1.3.3.2 Representation of States and the Planning Network

Any system for doing problem solving needs to be able to work with representations of domain worlds (often termed "world states") and representations of problem solving states. An example of a world state in MOLGEN is a sample which might contain a variety of hypothesized DNA structures, enzymes, and other reagents. An example of a problem solving state is a sequence of laboratory steps and world states which represent several steps of an experiment. We find it convenient to consider a problem solving state as having levels of detail, with different kinds of problem solving information on the different levels. A description of our approach using such levels is given in [36], section V.3.

Two main ideas have influenced our design. First, the world states and planning states are fairly complicated structures. It is necessary for a planning program to communicate with its users about its current planning state. In particular, it must be able to display its progress in an easily understandable form and it must be able to integrate suggestions from the user to alter its course. Rather than create separate programs for doing this, we have decided to represent the planning and world state knowledge using units in the knowledge base.

The second idea is that the process of problem solving can be expressed adequately by a small number of basic operations. These operations are used repeatedly in the course of solving a complicated problem. This is discussed in the next section.

1.3.3.3 An Eclectic Perspective on Problem Solving -- The AI Toolbox

When we study the solution to a problem in an unfamiliar domain, the first reaction is to be overwhelmed by the new detail and terminology. When one has mastered the terminology, the problem solving process can be viewed in better perspective. To be sure, many solutions remain brilliant and surprising. The majority of the solutions are easy to follow and we may recognize the solution process in the mind of the problem solver. Here he is sketching out some goals; now he is selecting an operator; now he is refining a step; now he has factored out a subproblem.

We believe that the process of problem solving can be modelled by a small number of standard operations. These operations include such things as (1) Sketching out planning islands; (2) Proposing subproblems; (3) Testing for mismatch of goal states; (4) Focusing attention on part of the problem; (5) Assigning time sequence to steps; (6) Selecting among competing choices for a given step; and (7) Splitting a problem into cases. Although these have been recognized as basic components of strategy, no existing system has integrated this breadth of strategy knowledge into a knowledge base. Dershowitz and Manna [9] have suggested the use of program schemata to represent general problem solving techniques such as Divide and Conquer. These program schemata are instantiated by the program synthesis system to obtain concrete programs. In our system the geneticist and/or programmer will create the instantiation. A major component of this research will be in the construction of such program schemata and instantiations. The package created will be termed the "AI toolbox". The creation of a library of schemata has been suggested by Gerhart [19]. The package will be built and tested in the context of a small number of genetic experiments. Computer science experiments with the package will include a measure of its performance and utility for expressing appropriate strategy for experiments.

Of course, having a number of basic tools does not guarantee that a program will use them correctly. Competence in problem solving in the domain requires that the expert system know where and when to apply the standard methods. Two factors are of importance here. (1) Some tools govern the use of other tools. For example, "Focus processes" will determine where to concentrate effort in a problem and indirectly control the selection of strategies. (2) The specific knowledge for using a tool is precisely the type of information which is left unspecified in the prototype and which must be supplied when concrete instances are acquired for the knowledge base.

The AI toolbox is discussed further in [36] in section V.4.

1.3.3.4 Simplifying Process Specifications by Removing Exceptions

Planning can be plagued by exceptional cases. If high level planning processes are burdened with the detail of the special cases, they can become cumbersome to update and debug. We are developing an object-centered approach, termed inspectors, for distributing the information about special cases throughout the knowledge base. This should help keep the planning processes clear and concise and also provide localized packets of information about the exceptions.

The basic ideas of this process of removing exceptions can be illustrated by an example of a "selection process" in designing an experiment. The operation of selecting among available laboratory steps is a recurring operation during experiment design. The following bear on this process:

1. The experimental goals -- e.g. to extract a section of a molecule.

2. World State specification -- a description of the laboratory sample, e.g. DNA structures.

3. The selection criteria -- e.g. availability, sensitivity, or functionality of the laboratory technique.

4. Verification criteria -- test for deciding after a simulation of the selected laboratory step whether the essential goals have been satisfied.

5. Failure instructions -- what to do if the chosen laboratory technique does not satisfy the goals.

6. Laboratory step specifications -- a list of the potentially applicable laboratory tools and their descriptions.

Our approach to managing the information about a selection process is first to identify the general information and then to consider the alternatives for placing information about the special cases. For example, section V.4.4 of [36] considers the selection of an enzyme to make cuts around a region in a DNA molecule so that it may be extracted. The general information in this case is that the main determinants of selection are enzyme availability and information about where it will make cuts. Special case information includes modifications on this basic idea according to unusual variations in the structure of the molecule -- e.g. "AT-rich regions", "hairpin loops", etc. It also includes information specific to the laboratory steps being selected -- e.g. nuclease contamination in an enzyme.

Interactions between goals often arise in the special cases (3). When action was taken to satisfy the preconditions of the enzyme (in this case, pH was changed), a side effect resulting from interaction with an unusual molecular feature interfered with the action of the enzyme. Specific advice about failures of this kind can be associated with the "pH inspector" -- To avoid conflicts of this kind an enzyme should be used at a suboptimal pH. Inspectors may be viewed as domain specific versions of planning critics as developed by Sussman [37] and Sacerdoti [32].

Generally there are many tradeoffs involved in deciding where to locate the information about special cases. The example above and the tradeoffs are described in detail in section V.4.4 of [36].

1.3.3.5 Further Aspects of Knowledge Base Management

One further area of work in knowledge base management that we will be pursuing is in developing a number of modest aids for a user for keeping track of a growing and evolving knowledge base. As discussed previously, we have already developed some facilities for automatic documentation of the knowledge base. We will be designing aids for a user in tracking down unexpected conflicts. For example, if two geneticists sharing a knowledge base have a somewhat different view of some aspect of the domain, one may make changes effecting the other's area. In the event of a failure, it would be useful in many occasions to get a summary of recent changes to the knowledge base. When a change is contemplated to the definition of some class of enzymes, it would be a simple matter to locate all of the rules which mention those enzymes. These rather simple aids are expected to be useful when fairly extensive changes to the knowledge base are contemplated because they will assist the user in being thorough about making his changes.

1.3.4 A Method for Designing Discrimination Experiments

One of the applications of the genetics knowledge base will be the building of a high performance system for designing a variety of discrimination/analysis type experiments. The goal in these experiments is to learn something about a given sample of DNA structures. For example, are any poly A regions present, or do the structures carry tetracycline resistance? The basic method used for the experiment design system will be means-ends analysis combined with hierarchical planning as follows:

---

(3) See Section III.2.5 of [36] for a discussion of recent techniques for handling interactions between goals.

1. A model structure containing the hypothesized feature(s) will be compared with a structure representing generalized DNA of the same type in order to ascertain exact differences.

2. These differences will be ordered and one selected as a basis for initial planning.

3. An experimental strategy, ranging from very specific (e.g. if a bubble is present then denature and use EM) to very general (e.g. label the feature and then look for the label) will be selected from a library of such "skeletal plans," using the difference selected.

4. The skeletal plan will be adapted to the specific problem environment, with hierarchical planning proceeding as deeply or shallowly as is desired by the system user.

5. The completed design will be tested in a forward direction for completeness and consistency by an evaluation system.

6. If a successful design cannot be found from the feature selected in (2), then either a new feature will be selected from the difference list, or a generalization selected from a tree of structural features will be used. For example, if the selected feature was the exact base sequence ATTGC, a generalization might be made to "known base sequence."

The following major components are needed for this experiment design method:

1. A problem analysis preprocessor which recognizes key features in the nucleic acid structures, nicks, gaps, hairpins and the like, and then compares these structural features to find major differences between candidates in a discrimination experiment. This program will organize features into a hierarchy of importance for experiment design using a rule-based system for analysis and classification.

2. A tree structure for ordering structural features and providing links between features and the classes to which they belong. The links will point upward as generality links to a more general class or downward as specificity links to a more specific one. The highest links will be to the complete class of

structural features, the lowest ones to individual specific features. Intermediate levels will consist of important subclasses, e.g. "poly base sequences" which would point downward to "poly A," "poly T," "poly G," and "poly C," and upward to "AT/GC ratio", which would itself point upward to known base sequences." This tree will provide entry into genetic strategies classified by the features to which they specifically pertain. Failure to find knowledge about what to do with a specific feature will cause the system to search for knowledge about the subclass of features immediately above the specific feature in the tree.

3. A cohesive system for hierarchical planning in the domain of molecular genetics--selecting strategies as described above and refining them downward to specific laboratory tools. This of course involves the usual need for error recovery and backtracking facilities.

4. A system for evaluating completed designs at any level of generality to be sure the plan as a whole "fits" togethers, i.e. a forward-working system for plan evaluation.

These components will be integrated into the basic MOLGEN representations framework of the Units system described above. The rules describing the problem analysis preprocessor will be individual units, as will the descriptions of structural features which combine to form the tree structure of the feature hierarchy. Individual dynamic planning states will be represented as units within the MOLGEN system.

After the experiment design system becomes operational for discrimination/analysis type experiments, we intend to adapt it to experiment-planning for synthesis experiments where the goal is to produce some desired DNA structure from available starting materials.

1.4 An Example of the Genetic Utility of Automated Experiment Design

A good way to illustrate the potential utility of computer assistance in experiment design is to show how some recently published work from another laboratory might have been represented in a MOLGEN formulation, albeit human intelligence was the instrument. The work in question achieved the cloning of the gene for rat insulin in a bacterial plasmid vector [38]. The major goal of the experiment was the transfer of a gene coding for insulin from the rat to the common

intestinal bacterium, E. coli. An important subtask of the experiment was: given samples of two different linear DNA structures with "sticky" ends, produce a circular structure containing one molecule of each. The difficulty of this problem lies in the number of competing processes. Both structures can self-circularize, and many different linear and circular monomers can be produced. See Appendix III for further details.

Previous attempts to cope with the problem were based the kinetic theory of ligation [11]. Using a model of the process based on concentration and molecular weight of the structures, one varied various experimental parameters to maximize the amount of the desired product. The new idea of the article was based on a different strategy--try to eliminate competing processes. This led to a method for modifying the sticky ends of the two structures so that they could no longer self-circularize. The particular method chosen was a simple biochemical step. The solution was retrospectively self-evident, but in fact, it was missed by many geneticists who had previously examined the problem (or related others). An intelligent experiment design system with the above mentioned heuristic probably would not have overlooked the solution.

2 Research Plans

The bulk of this application comprises details of research strategies for the first two years of the renewal period. It is of course more difficult to forecast over longer periods; indeed there is every likelihood that unforeseen difficulties and opportunities will intervene to offer changes of perspective. However, enough progress may have been made to enable us to move from the initial stepping stones, and our current plan for years 3 and 4 is as follows:

We will by then have invested substantial effort (mainly through the cooperation of investigators with direct support for molecular genetics research) in building and maintaining the knowledge base in the specified domain. We believe that this investment should be exploited before substantial further efforts are made to expand the domain -- e.g. in more biological aspects of genetics -- although a few easy opportunities will surely present themselves. Instead our emphasis will be on the evolution of MOLGEN to a hypothesis-oriented system (like DENDRAL). Typical questions at that level would be: from the data given, what are plausible hypotheses for the structure of a sample of DNA; as well as, what experimental steps should be pursued to verify the hypothesis. Intermediate steps have already been mentioned: the elaboration of hypotheses in the course of debugging, and the extension of MOLGEN from a sharp binary discrimination (between two

stated alternatives) to a corroborative mode (a given hypothesis versus all plausible alternatives).

This effort will require a good deal of work on the planning components and on rules of plausible induction, and relatively little on the knowledge base of molecular genetics per se. For that reason it should have the greater value for extensions to other domains.

2.1 Building and Maintaining the Genetics Knowledge Base

One important building block for MOLGEN's success is the creation and updating of a base of knowledge about genetics. This provides both a useful reference source for the user planning his own experiments, and a body of core data on which automated experiment planning strategies can be tested and evaluated.

The knowledge acquisition and representation ideas have been delineated in detail previously. This section describes advantages of the resources available here to support this work.

The strength of the Stanford community of biologists, biochemists, and geneticists offers a unique opportunity for collaboration in building a substantial body of knowledge about on-going research projects. Many of these projects may prove to be suitable sub-domains for initial development of the knowledge base, and may provide test domains for work on problems of knowledge acquisition and knowledge base management. They offer the added advantages of insuring that this work is solidly grounded in real-world experiments.

Updating and checking the knowledge base with respect to the information obtained from our collaborators will be the responsibility of the graduate student and post-doctoral fellow in genetics. On oppornunistic occasions they will also experimentally validate those experiment plans generated by the system which are relevant to current work in the lab. This has already been done in the case of ligation kinetics, where theoretical predictions of plasmid self-circularization were compared to electron-microscopic empirical tests (Section 1.2.2.1).

One useful side-effect of this process that we anticipate is that the process of formalizing knowledge about the domain may help to organize what is currently an informal body of knowledge, and in doing so may even uncover gaps in our current store of knowledge about the field.

2.2 Recognizing and Abstracting and Saving Successful Plans

As noted above, our efforts to create an experiment-design system center around the concept of a large knowledge base containing task-specific information. One approach to augmenting this knowledge base is via interactive knowledge acquisition as discussed in Section 1.2.1.3. A second form of knowledge-base improvement is based on giving the system the ability to save successful experiment designs it has generated. This involves two major functions: recognizing when a plan is worth saving, and abstracting a plan so that it is applicable to a wider range of problems than just the specific one which prompted its creation.

For example, consider the design MOLGEN might produce for the problem of ligating two genes. An initial attempt would be made to produce "sticky" ends by cutting both genes with a single restriction enzyme. Suppose, however, that no restriction enzyme which satisfied this criteria could be found. One possible solution would be to cut the DNA near one gene with restriction enzyme A, and near the other with a different restriction enzyme B, then join the two segments by means of a small piece of artificially created DNA which had been cut on one end by A and the other by B. This general idea -- the concept of a "molecular adapter" [35] -- is very useful for problems of this sort. While recognizing and abstracting the relevant ideas from the specific experiment design above is difficult, the ability to do this would be an important form of knowledge base augmentation.

Automating the function of recognizing when a plan should be saved will rely on several interrelated factors:

1. Was the plan a "good" one, i.e. does it solve a problem with reasonable efficiency, cost, safety, etc.? This measure of "goodness" will be difficult for the system to judge alone. The system will rank alternate plans according to its heuristics which involve these factors, but an absolute measure of plan quality will be a very difficult measure. Until the knowledge base includes adequate metrics for the "goodness" of plans in a variety of subfields, we imagine this judgment will be mostly up to the user.

2. If the plan solves a problem for which other plans already exist in the knowledge base, is the plan a significant improvement? Making this judgment involves use of the same "goodness" metrics discussed above as well as an analysis of why the new plan was not simply a copy of the old. If the only difference between the two plans is that the new one takes account of some detail of the environment for a specific problem, then it probably isn't worth saving. Again, we imagine this judgment will, at least initially, be mostly up to the user.

3. Is the cost of saving the plan less than the cost of regenerating it each time the same problem arises? Empirical criteria applicable to making the tradeoff decision might include CPU time spent in generating the plan, and, if the plan evolved directly from a previously saved one, the number of differences between the two plans. Another important measure is some judgment about how often the plan will be useful in the future. The more generally useful a plan, the more important it is to keep it around.

4. Can the plan be abstracted to more general purposes? In the example given above, this would involve recognizing that the concept of a "molecular adapter" would be useful any time the goal was to join two pieces of DNA which did not have convenient restriction sites in common.

The problem of abstracting a plan that has been selected for saving has been considered before, in the robot planning domain, in the work on MACROPS in the STRIPS system [15]. STRIPS generalized successful plans in the following manner. Plans had variables which were bound to objects in the robot world. The idea was to "unbind" these variables as much as possible, with certain constraints imposed by the nature of the particular plan. For example, if one step of a plan told the robot to go to a block and the next step said pick up a block, the block in these two steps must be the same one. Initial references to specific doors, blocks, rooms, etc. were generalized to "any door," "any block," "any room,", etc., but took into account constraints of the sort noted above.

We will begin by employing much the same generalization process. For the example plan given above, the specific restriction enzymes could be generalized to any two distinct restriction enzymes, and the two genes to any two DNA sequences lacking a common restriction site. This generalization process should be a natural consequence of the hierarchical nature of our knowledge base. The taxonomical classification of DNA structural features discussed above (Section 1.3.4) could be used to generalize plan utility--e.g. if a plan solves a problem for nicks, maybe it can be generalized to solving that problem for the immediately more general parent of nicks, DNA excisions. A plan which was useful for recognizing a specific base sequence might be generalized to one which was useful for all "known base sequences." The generalization of variables within a plan will be done by moving up generality links (see Section 1.2.1.1) in the knowledge base. A particular exonuclease could be parameterized to mean any exonuclease--the parent of all specific exonucleases in the knowledge base.

One difficulty lies in knowing how far it is possible to go up the generality links in the knowledge base before losing plan utility. A specific enzyme could be unbound to refer to "any enzyme," but that

would yield a plan which said "pick any enzyme", and would probably be far too general. The problem lies in detecting the important (i.e. important to this particular plan) features of each object used in the plan, and trying to retain those features while generalizing out all "irrelevant details." In the plan discussed above, the important feature of the chosen restriction enzymes is that they cut at specific sites, not that they were derived from some particular organism or that they operate optimally at a certain temperature. Note that such information (the degree of relevance of the features) will be available from the hierarchical planning phase, since it will have been used to choose among competing laboratory "tools". In this case, for example, restriction enzymes were chosen because they cut at specific sites. So, in unbinding the notion of a particular restriction enzyme, if the knowledge of the importance of specificity is maintained, progress up the generality links will end at "restriction enzyme."

2.2.1 A Casebook of Unsolved Problems

A related area of investigation -- recognizing novel combinations of laboratory techniques -- is based on the observation that new tools are continuously being developed. Sometimes a particularly useful combination of tools is available for quite some time before it is recognized as such, as in the use of alkaline phosphatase to inhibit self-ligation mentioned in Section 1.4.

One approach for doing this is to keep on file a casebook of important or unsolved genetic problems. Periodically as new laboratory techniques and planning strategies are added to the knowledge base, we will run the planning program on the test cases.

The process which we would like to model is embodied in the situation where a scientist, after hearing of a new laboratory technique, recognizes a laboratory problem for which the technique could be profitably applied. Thus, the first step of the process is to select a problem (from the casebook) after being presented with a new technique. Then the existing planning methods in the system would be applied on the problem again with the new laboratory technique encoded in the knowledge base. The next phase is deciding whether a new solution using the new technique offers any advantage over previous solutions. This would make use of the work described in Section 2.2 for abstracting and evaluating plans.

In summary, this process will make use of other developments in the MOLGEN project for applying newly discovered techniques and for evaluating the plans produced. These methods will be applied to form a discovery process by augmenting them with a casebook of experiments and a selection process for picking experiments.

2.3 Understanding Experimental Discrepancies by Hypothesis Formation

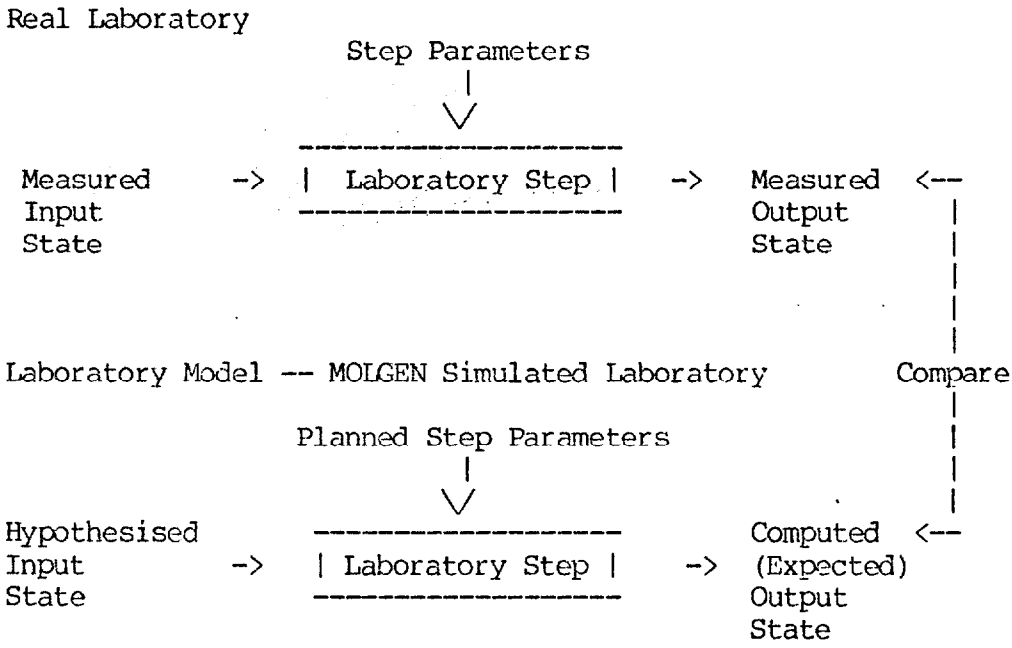
Plans for molecular genetics experiments have something in common with computer programs, summer vacation trips, and almost any kind of plan devised -- they don't always work. Although published reports about experiments usually say little about the techniques which failed, our analysis of the actual development of experiments has shown us that debugging is an essential and integral part of successful laboratory experimentation [14].

Pioneering work in the debugging of computer programs [37] used a process of program development and correction using a knowledge base about bugs. Sussman has advocated creating a rough version of a program on a first pass followed by local corrections by a gallery of critics. In Sussman's work, the ignoring of detail during the first pass is a source of power for the approach. In experiment debugging, we find an additional reason for attending to details during a second pass -- the knowledge necessary for deciding which of the details are in difficulty is not available at the time the experiment is designed. For example, many assumptions about the input samples are made when the experiment is designed and the validity of these assumptions cannot practically be tested until the experiment is performed.

Section 2.3.1 describes several distinct sources of error in experiment design. Section 2.3.4 describes a number of research issues.

2.3.1 Sources of Bugs

The complexity of most laboratory techniques is such that there are many ways for an experiment to go awry. The following diagram will be used to categorize the sources of bugs.



In this diagram, we assume that a complete plan for a laboratory experiment has been developed. In this diagram, we will focus on a single step in the experiment. The upper part of the diagram corresponds to the step as carried out in the laboratory -- illustrating that a laboratory sample undergoes processing in a laboratory step resulting in a changed sample. Measurements are made on both the input and output samples. (4)

The lower part of the diagram corresponds to a model of the laboratory process. This may be either an informal model in the experimenter's thoughts or a formal model in the MOLGEN knowledge base. The model transforms a hypothesized version of the sample using a process model of the laboratory step to yield a model of the expected results of the laboratory step.

If at some point in an experiment the measurements predicted by the model fail to correspond to actual measurements in the laboratory, a "bug" has been encountered in the experiment. Referring back to the diagram, the source of the bug may be any of the following:

1. The hypothesized input sample may be inaccurate.

2. The planned parameters for the laboratory step might be not correspond to the actual parameters used.

---

(4) Measurements are examples of laboratory steps, subject to the same descriptions and errors as other laboratory steps. This diagram has been simplified by leaving implicit the measurement step.

3. The process model for the laboratory step could be inaccurate. This inaccuracy could be in either the laboratory step shown explicitly or in one of the implicit measurement steps.

4. The error may be traceable to any of the above sources in any of the previous steps in the experiment.

2.3.2 Knowledge about Debugging

A Molgen debugging system must be able to generate and test hypotheses about bugs from any of the sources mentioned above.

The first source of bugs mentioned above -- an inaccurate hypothesis about an input state -- is a common source of difficulty in experiments. Knowledge about DNA structures, a major part of an input state in most experiments, is almost never complete. For example, there is a danger of minor damage to DNA structures in almost any laboratory step so that nicks, gaps, and various forms of erosion tend to appear in these structures in the course of an experiment. Whereas experiment planning derives much of its power by ignoring such details, experiment debugging is likely to derive its power by scrupulous attention to these possibilities when problems arise. A discrepancy in the parameters of a laboratory step is similar in principle to a discrepancy in the input state. There are always many possible sources of error here: Was the pipette sterile? Were there nuclease contaminants? Was the pH controlled properly in the buffer? If a simple error of this kind can explain the discrepancy in the experiment, there is probably little need to look farther.

A more difficult source of error occurs when the model of the laboratory process is inaccurate or incomplete. Since most of the debugging which we have observed involves the other sources of bugs, we expect to put most of our effort there. We have, however, some modest ideas using analogical reasoning for extending and correcting process models. Suppose that we suspect that knowledge about enzyme A is incomplete, and suppose also that we know that enzyme A is similar to enzyme B (a well-understood enzyme). If the operation of enzyme B depends on the concentration of a magnesium ion and the model for enzyme A does not mention this ion, a reasonable question in the face of a failure in using enzyme A might be whether the model for enzyme A is deficient in this aspect. While very simple, this approach may prove to be adequate to provide useful hints.

2.3.3 A Debugging Example

This section gives an example of debugging a step in a genetics

experiment. (5) The example illustrates two models of genetic laboratory steps and shows the generation and testing of two hypotheses about a bug. After the fairly lengthly example of debugging which follows, the research issues involved in this approach are presented in Section 2.3.4.

2.3.3.1 Generating Debugging Hypotheses

As stated above, experiment design must usually be done with incomplete knowledge. In this example, the sample consisted of uniform DNA molecules from a bacteriophage. A laboratory step was proposed to cut the DNA molecules into smaller segments so that a desired gene (Thy) would be located on a smaller segment for later manipulation. A restriction enzyme (EcoRI) was chosen on the basis of availability in the laboratory to perform the cutting. It was not known (a) whether the Thy gene had a restriction site for EcoRI (i.e. whether the enzyme would cut the gene) or (b) what the size would be of the segment carrying the Thy gene. It was essential for later steps in the experiment that the Thy gene be located intact on an appropriately sized segment of DNA.

To test whether the Thy gene was functional after cutting the DNA with the enzyme, a checkpoint was used involving "transformation". Transformation is a process by which bacteria can incorporate DNA from their growth medium. In this case, conditions were established so that the bacteria could survive only if they were transformed by a functional Thy gene. The bacterium used in the transformation test was Thy- B. subtilis. Lacking the ability to synthesize thymidine itself, this bacterium can normally grow only if the medium supplies the thymidine. During transformation, if bacteria can incorporate a piece of the digested DNA carrying the Thy gene, the "transformed" B. subtilis will be able to grow on a medium lacking thymidine.

The uncut bacteriophage DNA is capable of transforming B. subtilis in this manner, and it was assumed that the cut DNA would function similarly. The unexpected observation (bug) was that although the cut DNA was indeed capable of transforming the B. subtilis, it did so with a greatly impaired efficiency. A fix was needed or the subsequent experimental steps could not be performed.

At this point, two hypotheses were suggested to explain the low efficiency.

1. The EcoRI enzyme cut the gene -- damaging its transforming ability.

---

(5) The example is from the beginning of the experiment described in [14].

2. Transforming activity decreases if the DNA segments are too small.

The first hypothesis derives from fact that a gene which has been modified will function with an impaired efficiency. In this scenario, if the B. subtilis has been transformed with a damaged Thy gene, it will still grow on a thymidine deficient medium, but not as effectively. The Thy gene could be damaged if it has an EcoRI site.

The second hypothesis was suggested from the fact that the length of DNA fragments (and other structural features) are known to influence the transformation process (6). In this scenario, if the segment containing the Thy gene is too short, some part of the transformation mechanism would operate less effectively and the gene would fail to be incorporated.

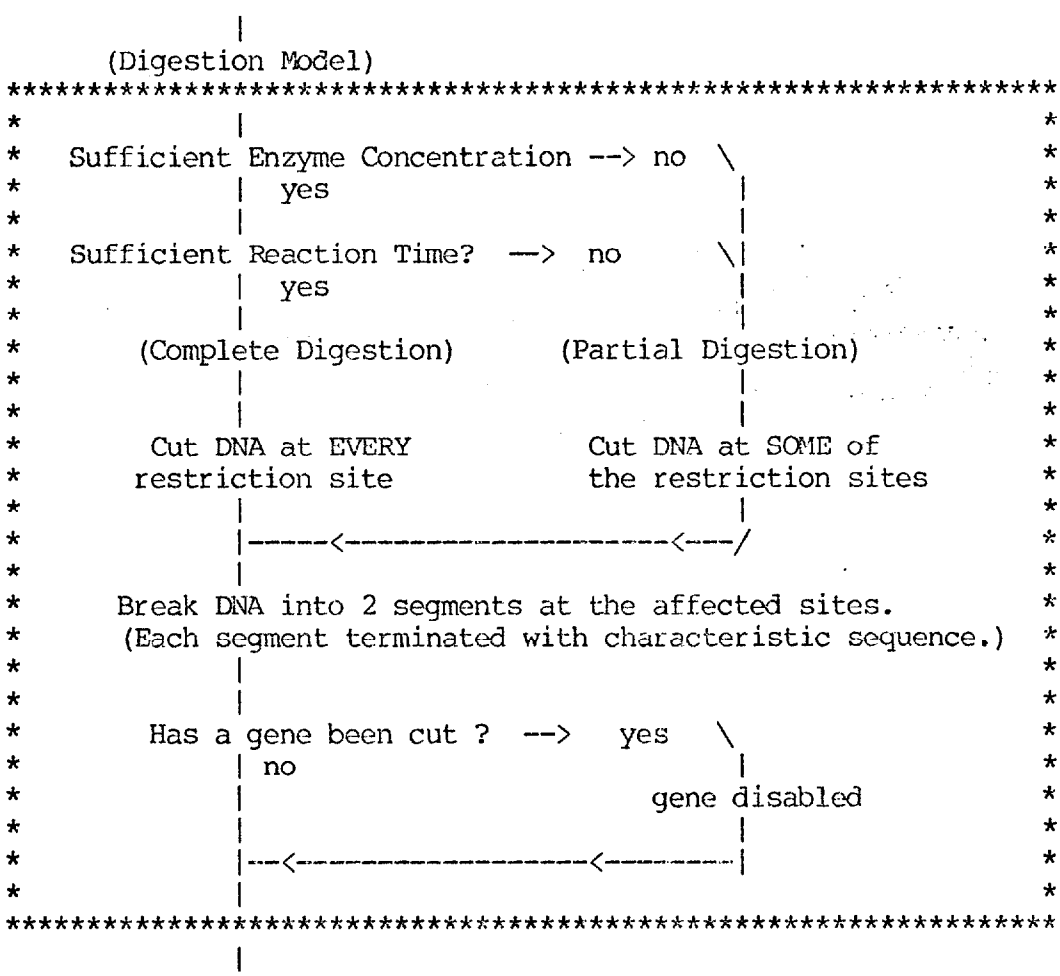
Both of these hypotheses may be derived by analysis of the process models for digestion (7) and transformation. These models are given below.

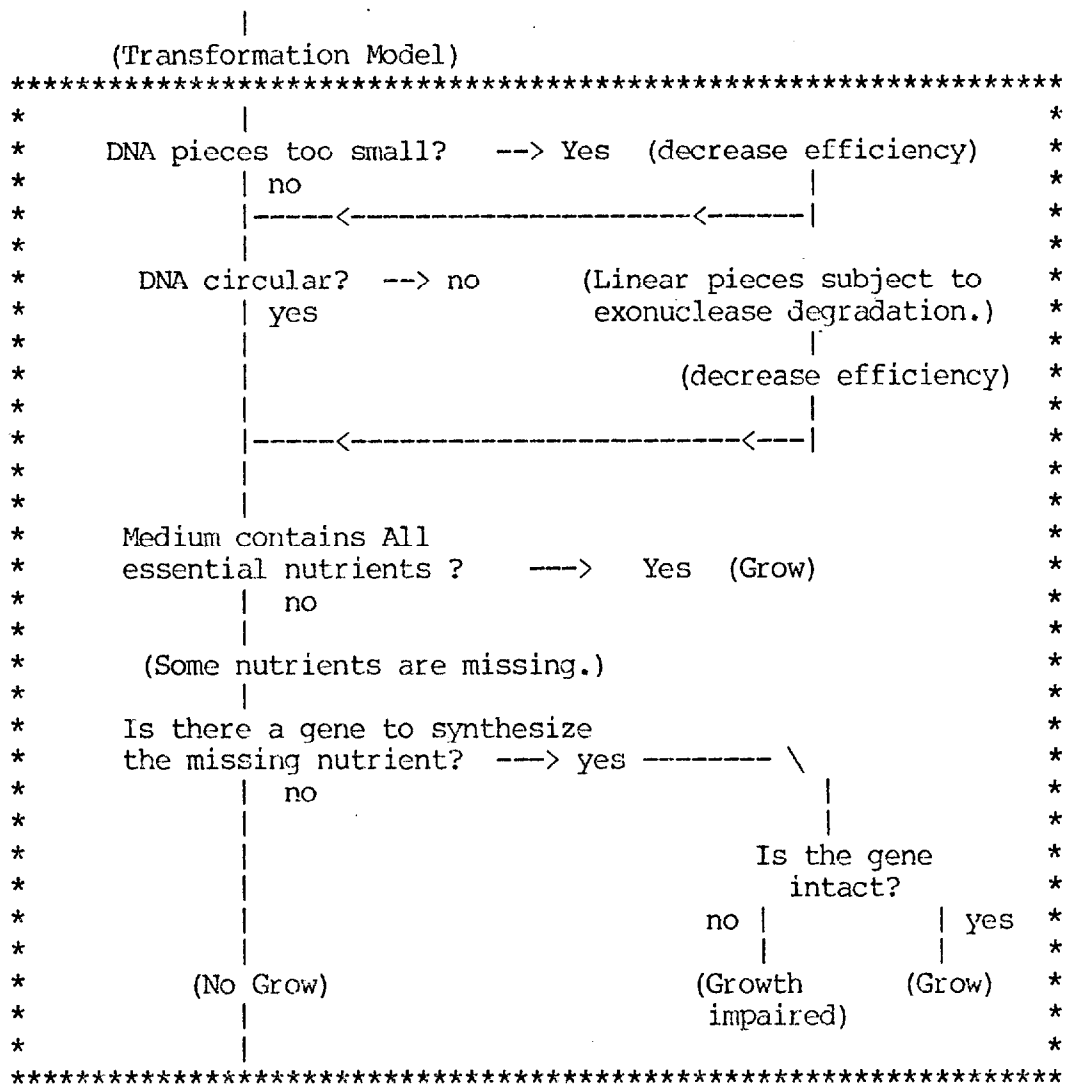
---

(6) See [28] for a detailed discussion of transformation.

(7) "Digestion" is a technical term referring to the cutting action of an enzyme.

Partial Models for Digestion and Transformation





2.3.3.2 Testing the Debugging Hypotheses

Once a hypothesis has been offered to explain a discrepancy in an experiment, it is often possible to propose a measurement which can validate it. If the hypothesized change to the model of the input state is adequate to account for the discrepancy in the output state, that is suggestive that the source of the bug has been found. Other times, a hypothesis about molecular damage may serve to suggest an alteration in the experiment or a way to fix the damage.

If more than one hypothesis for the cause of failure is found, it is necessary to determine which of the bugs is the most likely. In

this experiment the following knowledge was used in discounting the first hypothesized bug:

> Test for restriction site:
>
> If a gene can be shown to be functional at all after its DNA has been completely digested by a restriction enzyme, the enzyme probably does not cut the gene.
>
> Test for Completeness of Digestion:
>
> If no change is observed in the restriction pattern (in electrophoresis) for DNA after a ten-fold increase in digestion time and enzyme concentration, the DNA may be assumed to be digested to completion. (8)

The completeness of digestion was demonstrated in the manner suggested, indicating that, since there was a measurable amount of transformation activity, the difficulty lay in the size of the fragments after Eco RI digestion.

Attention now goes to the process of finding a fix for the problem. This phase of the debugging process is essentially an application of the experiment planning part of MOLGEN. In the current experiment, methods were sought which would allow the DNA to be cleaved, but in such a manner that the segment containing the Thy gene would be left on a larger fragment. Two possibilities were considered for this -- selection of an alternate restriction enzyme and a "partial" digestion in place of a complete digestion by EcoRI. The models which were used for generating the bug hypotheses may also be used to generate the alternative of partial digestion. In particular we can derive from the models that (1) the average piece will be larger if digestion is partial instead of complete, and (2) this may enhance the transformation step.

2.3.4 Research issues

The approach to debugging illustrated above relies on the use

---

(8) Although this rule was cited in this form in [12], some exceptions to it are generally recognized. In the first place, there are inherent resolution limits which prevent some changes in restriction patterns from being observable in electrophoresis. Secondly, sometimes restriction sites can be covered by trace proteins.

of checkpoints in an experiment. The judicious selection of checkpoints is an important aspect of experiments and illustrates that some of the interplay between experiment designing and debugging can be anticipated. The checkpoint above indicated that no further steps should be taken in the experiment until the check (transforming capability) was passed. Experiment design involves the use of other kinds of checkpoints -- e.g. "controls" which serve as checks on experimental artifacts. Selection and placement of checkpoints in experiments depends on many things -- e.g. the time and expense and sensitivity of the check. We anticipate that the development of the debugging system will lead to a number of additions to the experiment design system for handling the selection and placement of checkpoints.

The example and discussion above have illustrated some major research issues involving the generation and testing of potential bugs. One issue is how should the list of possible "bugs" be pruned when it is too large to test experimentally. Several sources of bugs have been illustrated above. A means must be found for focusing attention, i.e. determining which are the most plausible sources of error. Representation of the debugging process will probably involve considerable expansion of the knowledge base. In addition, the models for debugging seem to involve use of more detailed knowledge than is available for experiment planning -- implying some extensions to our modeling of genetic objects and processes as well.

Finally, one research issue lies in the source and organization of the models, such as those for digestion and transformation, illustrated above. Initially, these will be hand-crafted entities whose contents will be dictated by the range of experiment bugs encountered. Eventually, however, we hope to be able to exploit the commonalities arising in several models, to develop a more basic encoding of the information. For example, both digestion and ligation could be characterized as instances of "chemical reactions". Stored with this concept would be an indication that "degree of completeness" was a relevant parameter in describing its outcome. This eliminates the necessity of encoding such information redundantly, and provides a more intuitively appealing organization.

2.4 Reasoning by Analogy

At the end of the current grant period, MOLGEN will be a working planning system. The knowledge base will include a library of process schema expressing problem solving techniques and domain operations, and a library of skeletal plans expressing general solutions to particular problems. The planning system will have techniques for hierarchical problem solving. This system provides an excellent environment for exploring various types of analogical reasoning.

There is a long intellectual tradition in philosophy of use and analysis of the concept of analogy. The tradition begins as early as Plato and Aristotle. In fact, some of the most famous and memorable passages in Plato's Dialogues hinge upon a brilliant and imaginative use of analogies. We have in mind especially the famous image of man in the caves able to observe only shadow as an analogy to the problem of perceiving the true character of eternal forms.

During the long and technical phase of philosophy identified with the Middle Ages, the concept of analogy achieved a central importance as part of the extensive discussions of analyzing and knowing the properties of God. The conceptual difficulty was the recognition that it is not necessarily cognitively possible for human beings, themselves, to know the unbounded powers, knowledge, etc., of the deity. Consequently there developed a long tradition of attempting to argue by analogy from known properties of humans to what should be the properties of an omniscient and omnipotent deity. This conceptual literature on analogy is still a vigorous one, and publications can be found as recent as the last decade.

Another stream of thought that has given a good deal of attention to the concept of analogy is the philosophical analysis of the problem of induction. A distinguished and diverse set of philosophers, including Hume, Kant, and John Stuart Mill, have all had important things to say about reasoning by analogy.

The development of an analogy requires the specification of the analogous concepts and a description of the manner in which the concepts are analogous. Once enough of a description is given to establish the analogy, the description is extended to derive new attributes of one of the concepts. Unfortunately, philosophers and psychologists have not given precise descriptions of the process of analogical reasoning. Precision is necessary in order to use analogical reasoning as a component of a computer problem solving system. Polya began a new approach to analogy in his book, Induction and Analogy in Mathematics. His examples and informal discussions have served as a useful starting place for recent computer science investigations. The computer science work of Evans [13], Kling [23], and Brown [4] are major contributions to the effort of defining analogy precisely .

The aspects of analogical reasoning we intend to explore within the context of MOLGEN include:

(1) Given a new problem specification and a set of problem specifications how can a problem analogous to the new problem be selected from the set.

(2) Given a problem specification and an analogous

problem specification for which there is a known solution, how can the analogy be used to solve the new problem.

(3) What representations of problems and solutions are facilitative for analogical reasoning (that is, for solving problems 1 and 2)?

(4) Problems 1, 2, and 3 viewed from the planning environment: given a planning situation and a library of planning processes, find and instantiate an analogous schema from the library to build a planning process applicable to the given situation.

(5) Integrate problem solving by analogy into the planning system as one of the possible tools for solving subproblems.

We will briefly discuss each of these aspects of analogical reasoning. Our purpose in this research is two-fold. Foremost, we want to analyze some of the current approaches to analogical reasoning in the context of a major problem solving system. Secondly, we want to add an analogical reasoning component to the production version of MOLGEN, hopefully extending the problem solving ability of the system.

2.4.1 Forming Analogies

Finding a problem analogous to a given problem could require an explosively large search. A purely syntactic approach to analogy formation which attempts to map the objects, concepts, functions and relations of one problem specification into another only reduces the search significantly if the problems are identical up to a set of substitutions. Even then, there may be a large number of mappings to test in order to discover the analogy. For this reason, we believe a knowledge of the semantics of the domain is essential to the establishment of analogies and their subsequent use in problem solving. This implies identifying the functional relationship of the concepts of the problem specification to the problem as a whole. These functional relationships must be preserved by any analogy mapping created. Brown [4] gives a systematic method for creating an analogy map based on syntax including the ability to map m-ary relations to m+k-ary relations for small k. A beginning attempt at identifying functional relationships has been made for genetic features in the context of binary discrimination problem specifications in Section 1.3.4. In this case, the implied functional relationship is that the identified feature is the only key concept in the problem. The analogy is either an identity or a generalization map on the feature.

This work will involve the creation of techniques for recognizing functional relationships and using them to restrict the search for an analogy map. It should be noted that this process of discovering an analogous problem does not require any specific technique for using the analogy to solve the original problem. However, a measure of how "good" an analogy is will be necessary.

2.4.2 Using Analogy to Solve Problems

Once two problems are claimed to be analogous, the way we use this information to solve the original problem depends on the nature of the analogy and the representation used to store the solution to the known problem. If the analogy map is an isomorphism such that the two problem solutions are identical up to a set of substitutions, then we can apply the analogy map to the solution of the stored problem to obtain a solution to the original problem. The map must be extended to include any concepts, functions, relations occuring in the solutions that did not occur in the problem description. More commonly, we will not obtain a correct solution by simply applying an extension of the analogy map to the stored solution. For example, the analogy may indicate that the problems have identical general plans, that they require the same change of representation or the same type of reasoning (see [23] for a general discussion of types of analogies). In all of these cases, the result of applying and extending the analogy map to the stored solution may not give a correct solution to the new problem. In the context of a general problem solving system there are several approaches to explore. The first approach uses the analogy to constrain the general problem solver. That is, instead of creating a solution by applying the analogy map, the analogy is used to limit the choice of representations and transformations which the problem solver can use. Another approach would set up new subproblems to be solved as the analogy is extended to the stored solution. If the subproblems can be solved, the result of applying the analogy map to the stored solution along with solutions to the subproblems gives an accurate solution to the original problem. A third approach is one suggested by Manna [9]. The analogy map is used to transform the stored solution into the incorrect new solution. Now the process is one of modifying and debugging the solution to satisfy the problem specification. (See [37] for a discussion of debugging techniques.)

Research problems include: Classification of the types of analogies for which each approach is best; A strategy for determining when to attempt the solution of a subproblem through analogy rather than the other problem solving techniques of the system; A comparative evaluation within the framework of a single system of the interactions between representations and the approaches listed above for obtaining correct solutions from the analogy. The next section describes several possible representations.

2.4.3 Representation of Solutions

Several different representations have been used in research on program synthesis by analogy including: program schemata [9]; concrete programs with input/output specification with/without a list of the transformations which produced the concrete program (unpublished report by R.Moll, University of Massachusetts and J. Ulrich, U. of New Mexico); concrete programs with associated intentional plans and justifications [4]. In this section we briefly indicate that the MOLGEN knowledge can be modified to represent solutions in similar manners.

2.4.3.1 Program Schemata

We have already indicated the relationship between program schemata and process schemata in Section 1.3.3.1. The program schema is a generalized version of the solution to the programming problem. We can also create generalized versions of the solutions to experiment design problems.

The MOLGEN knowledge base has many plans which are almost schemata, namely the skeletal plans of Section 1.3.4. and the right hand sides of refinement rules. Also, Section 2.2 proposes techniques for the identification of new skeletal plans and their addition to the knowledge base.

These plans can be extended to become program schema. We first will associate with each plan an input/output problem specification. We generate intermediate world state descriptions by running the plan on the input world state. A generalization process, starting with the I/O specifications and proceding to include each world state in the plan will create world state descriptions in terms of abstract symbols with restrictions on the substitutions allowed for the abstract symbols. This process will generate preconditions which restrict the refinement and specification of the generalized transformations specified by the original plan.

Representing stored solutions as program schemata could have two advantages. First, the myriad of details present in a primitive description of an experiment procedure are suppressed. Our intuition is that these details are dependent on the specific transformations used in the procedure and thus would interfere with the creation of the analogous solution. Second, the analogous solution could be verified at the abstract level of the program schema. The general problem solving system could refine the schema using the restrictions generated in forming the analogy. Thus analogy would be used to guide the problem solver quickly to a workable plan without having to consider details.

2.4.3.2 Solutions as Concrete Programs

There are many systems which create analogies from a concrete program with I/O specifications. We are not building a library of such experiment procedures in the present grant period. However such a library will be created in the renewal period for studying analogical reasoning. After completion of a planning run, the user can specify that the experiment procedure created is to be saved. Saving the planning transformations which created the solution is simple since the system keeps such a record in the planning network. Thus both representations can easily be used.

As explained above, it is our intuition that the detail of the primitive procedures (concrete programs) will interfere with the analogy formation. However, this intuition will be tested.

A different use of the analogy is intended when the planning transformations are stored with the solution. Now to create the analogous solution, one applies the transformations (modified by the analogy map where necessary) to the original problem specification. Again, the resulting solution may not be correct and thus may need a modification/debug cycle.

2.4.3.3 Programs/Intentional Plans/Justifications

Brown [4] has suggested that solutions (programs) should have three components: the code, an intentional plan, and a justification. There are transformations which create code from plan, justification from plan and so on. Each component is important in his analogy system. We will develop a verification, or justification, language for experiment procedures. At the present time MOLGEN does not include the use of intentional information in the description of a rule. However, the names of the generalized transformations have been serving this purpose. We suggest the intentional information be made an explicit part of the transformation's representation. This is particularly important in planning processes. It is not always possible to deduce the process intention from the process itself. For example, a planning process might check for two conditions which the process creator knew were indicative of a certain situation. If the situation itself were never mentioned in the process, the intention of the process creator could not always be deduced from the process. The experiment procedure representing the final product of the planning process could be annotated to indicate the planning program's intentions in creating the steps of the plan. If one considers the hierarchy of abstract transformations created during the planning process as a net indicating various levels of generalized transformations, then each level of the net specifies an intentional plan for the following level. Initially we will concentrate on creating intentional plans for primitive and

generalized transformations and program schemata. Then we will extend the concept to include all process units.

2.4.4 Creating Context Dependent Planning Processes

So far, we have been limiting the analogical reasoning to finding and using analogies between problem specifications. However, many of the same ideas can be applied to the design of a component which examines the planning state and the current goals, selects a planning process schema whose input/output specifications are analogous to the present planning context and creates an instantiation of the schema. The major contribution of this work would be the identification of the functional relationships among the planning concepts represented in the planning state. In fact, this search for analogous planning contexts turns one of the problem solving techniques back on the problem solving process itself and allows the system to modify its process knowledge base dynamically. We view this as a difficult problem which awaits the further specification of the planning state representation for more detailed approaches.

2.4.5 Integration of Analogical Reasoning and MOLGEN

We have indicated above that if our analysis of diverse methods of analogical reasoning shows one to be advantageous in the MOLGEN context, we would add this technique as a component of the production system. This involves the creation of process units describing application criteria for the analogy package. The analogy packages themselves will be designed within the framework of the current MOLGEN representation paradigm and thus will be compatible with the rest of the system. As with other general AI problem solving techniques, the main difficulty will be in accurately specifying the appropriate contexts for application. There will be some adjustment if the successful representation method is a variant of one currently in use. However, the extensions we have mentioned are additions to the information represented rather than major modifications of the representation itself.

2.5 Performance Evaluation and Improvement of AI Knowledge-Based Systems

Overview: Performance Evaluation as an AI Problem Solving Task

The objective of this part of the proposed research is to investigate methods of automating the process of measuring, evaluating, and improving MOLGEN's performance.

The work is motivated by the belief that, for AI systems that attempt to solve real world problems effectively, it is just as important to have a representation of how knowledge is used during problem solving, as it is to have representations of how that knowledge is organized. For example, inefficiencies in testing rules, excessive backtracking, too frequent or infrequent access of specific items in the knowledge base, adequate but consistently suboptimal answers to problems, may all be symptoms of a mismatch between the user's conception of the domain and the structure of the domain required for the efficient solution of problems under consideration. Performance evaluation tools can be used to detect these performance problems and, used properly, can give the user a clearer understanding of the domain. Guided by its measurements of performance, the system may be able to assist the user in proposing and testing changes to reflect this understanding. A sophisticated system would use its own performance statistics to propose changes that would better reflect current usage.

The work is also motivated by the conviction that performance knowledge is essential for conveying a program's scope and limitations to a user, who can then use the program more intelligently. Thus, a second motivation is to increase the user's knowledge of the system's capabilities. Experience with knowledge based systems has shown that one of the biggest investments of time and effort made by a starting user is in finding out the scope and limitations of the program. Buchanan and Smith [5] has an excellent discussion of this problem in the use of the CONGEN program [6]. CONGEN is a sophisticated generator of chemical structures under user-provided constraints. Users of CONGEN have found that manual assistance is insufficient to acquaint them with the sorts of constraints which the program can accept. In trying to make the system easier for the expert to use, it is also important where possible to give the user some idea of how much of the problem has been solved, how much remains to be done, and where the system has been spending most of its effort. CONGEN and SECS [40] let the user see current problem status by means of special user commands or dynamic displays of the problem solving graph. While not all of these measurements will be meaningful for every type of problem solving, giving the user at minimum a description of what the system is working most on will greatly increase understanding of how it attacks problems of different kinds. Thus, a basic goal of incorporating performance evaluation tools into AI problem solvers should be to provide descriptions of the distribution of system effort -- what information was accessed, what rules were invoked.

2.5.1 Creating a Knowledge Base for Performance Diagnosis and Correction

We take the point of view in this research that evaluating a system's performance and suggesting improvements is itself an AI

problem solving task. The diagnosis of system performance problems can be viewed as a problem of hypothesis formation and verification. Elementary actions of the system can be viewed as "signals" that must be interpreted and summarized to form patterns of behavior, or system "symptoms". Correction will be based on user definitions of what changes should be made in the system following the detection of specific symptoms.

An important part of the research will be to formulate knowledge about measurement, evaluation, and improvement in terms of rules. The rules in this "performance evaluation" knowledge base will perform a variety of functions. They will:

1. interpret elementary system events as hypotheses about system behavior.

2. initiate and focus additional measurements on the basis of proposed hypotheses about behavior.

3. suggest modifications in the organization of the system's knowledge base or in its operators and strategies as a consequence of a sufficiently confirmed diagnosis.

A sophisticated system should help the user define different sets of these rules for different system performance goals. That is, the user should be able to specify what behavior to measure and what modifications to make depending upon the importance which he assigns to various criteria of system performance. These criteria include efficiency, quality of answer, plausibility of processing sequence, and clarity of the knowledge base. As in medicine, there would be no explicit definition of "normal" system behavior. Instead, the user's assignment of performance priorities would effectively define "healthy" system behavior, by designating the circumstances under which modifications should be made (in other words, by defining "unhealthy" behavior).

The effects of a performance evaluation "meta system" acting upon a problem solving "object system" should be visible in several ways. The object system should work more efficiently, and on a wider range of problems, getting improved answers. The user should get feedback about how the system works, where it spends most of its effort, and how each new rule or object affects system performance. The guidance by the system of acquisition of new rules and objects from the user should reflect its experience with how existing rules and objects affect performance. Perhaps most important, the user should become better informed about the connections among various sources of knowledge in his own domain, at least for the set of problems confronted by the problem solving system.

This investigation will primarily be concerned with how well the knowledge base of an AI problem solving system is organized and used. The emphasis will be on such factors as accessing and grouping information about domain objects, and retrieving, testing, and invoking domain rules. There are, of course, other important determinants of system performance, such as the selection of the best internal representation for a data structure, or the selection of the most efficient method of searching a data file. However, the main concern here is to assist users in organizing domain knowledge for effective problem solving in conjunction with the AI system. Therefore, we will concentrate on the aspects of the system that might be modifiable directly by the domain expert.

Proposed Research Steps

The specific steps of the research will be:

1. Identify the elementary events of rule based problem solving systems which may affect performance.

2. Establish simple measurement techniques for detecting these events.

3. Develop interpretations of these events as higher level hypotheses about system behavior.

4. Formulate these interpretations as rules which map events into descriptions of behavior.

5. Formulate inverse mappings: rules that initiate additional measurements on the basis of behavior hypotheses suggested by existing data. Establish measurement methods as needed.

6. Identify what changes might be made to improve various kinds of performance.

7. Investigate how to use top level descriptions of behavior to suggest system modifications, and formulate these as modification rules.

It is hoped that these steps will have visible positive effects on system performance and user knowledgability. In addition, it is hoped that this research opens the way to: (1) expressing desired system performance in the form of sets of modification rules, and (2) using past evaluations to estimate the impact of new knowledge and to guide the acquisition process.

Performance Criteria

The whole point of measuring what the system is doing is to improve it with respect to established performance criteria. A major aim of the research is to make it easy for a user to express different criteria and as a result to get different sets of responses to object system symptoms. Here are some examples of performance criteria and how they might differ in the changes they recommend.

1. Get an adequate answer with a minimum of effort. The most important factor is efficiency.

2. Get the best answer and don't overlook any possibilities. This is a conservative system.

3. Keep the knowledge base as simple as possible. One motivation would be the ease of understanding how the system gets its answers.

4. Follow a sensible line of processing. Perhaps it is most important that the solution method resemble the designer's idea of how a human solves the problem.

Each of these criteria dominates the design of at least one major AI problem solving program. The eventual aim of this line of research should be to understand the behavior of AI problem solvers well enough so that, with system help, a user could specify precise performance criteria which the system would translate automatically into a set of measurement and correction rules. Short of this, it will still be helpful to give the user part of these capabilities. The user should be able to write correction rules, he should be given feedback on their effects, and he should be able to designate alternative sets of correction rules to correspond to different performance goals.

However much of the process of creating therapies is automated, the practical product of establishing criteria will be a set of rules giving correction recommendations for detected symptoms. Two brief examples will illustrate the differences in recommended changes when different criteria are in force:

1. An operator fails to achieve its desired effect fairly often. The choice is whether to add a check for the upsetting condition or to rely on backtracking. If the performance criterion is "efficiency", the recommendation might be to add the check. If the criterion is "simplest adequate knowledge base", where "simplest" includes fewest operator preconditions, the recommendation might be to leave the operator as is.

2. The criterion of "sensible order of processing" might emphasize pursuing a single search path as long as its

heuristic rating is above some minimum. By contrast, a "best answer" criterion might demand that the most promising node be expanded at each step. Operator selection strategies would surely be modified differently to meet these two criteria.

3 Significance

3.1 Significance to Computer Science

The proposed research brings together many themes of recent artificial intelligence work. The task area of molecular genetics is richer and more complex than other tasks in which these themes were originally developed. Therefore, we view MOLGEN as research on extensions of currently known methods as well as on integration and application of those methods.

In the Introduction we listed several of the specific issues we believe are critical for developing reasoning programs that aid scientists. We mention only in passing the fundamental issues of representing, managing, and acquiring knowledge for a reasoning program because we take these to be inescapable in AI research. While we have no radical discoveries in these areas, we have tried to state clearly in the proposal how we approach these "knowledge engineering" issues.

3.1.1 Reasoning by Abstraction

The most highly developed program to exploit several levels of abstraction in its reasoning [32] works in simple domains with few facts and relations. This pioneering work will be refined and extended by the work on MOLGEN so that reasoning in complex domains can be guided by scientific knowledge at many levels.

The basic knowledge with which MOLGEN solves problems has been organized from the start in a hierarchy that reflects successively more detailed descriptions of objects and operators. The proposed work on diagnosing failure in designs for experiments operates on the premise that one major cause of failure is using general knowledge when more detailed specifications are required. On the other hand, the programs that plan the experiments gain their power from omitting details.

We also propose to use the abstraction hierarchy to improve the knowledge base in light of experience. Again, the general descriptions

of successful experiments will be the ones MOLGEN will be able to apply to new problems, and the ones of most interest to researchers.

3.1.2 Strategy Knowledge

As the knowledge base of facts and inference rules increases in a program, it is necessary to find efficient means of reducing the amount of computation at every reasoning step. The most sophisticated AI technique for controlling a reasoning program is to encode and use strategy knowledge to guide the program into the most useful facts and rules, without considering the others.

The Teiresias system [7], developed at Stanford, demonstrates the power of encoding strategy knowledge in rules and using it to guide the invocation of domain specific knowledge. In MOLGEN, the need for strategies to guide the processes of designing experiments and diagnosing failures will be acute due to the amount of potentially relevant information at each step.

3.1.3 Integration of Diverse Sources of Knowledge

Reasoning about complex problems requires integrating information from more than one source. Problem solving is not neatly compartmentalized into independent packages of relevant material. On the contrary, expert problem solvers know how to use information from many sources about many different aspects of the problem.

The HEARSAY programs for speech understanding are among the best known examples of bringing multiple "experts" into a common problem solving process [22]. There, each expert contributes what it can to a current best hypothesis with little communication among the experts themselves.

In nearly every aspect of the MOLGEN program, there are several ways of viewing problems, each with its sources of information and problem solving procedures. We are therefore looking for general mechanisms that enable different experts to cooperate on problems of various kinds. For example, both the experiment planning and the debugging systems have to call on experts with knowledge of different instruments and experimental procedures. The experts may not have a common vocabulary, yet they must be able to contribute to the problem solving at different levels of abstraction and about different parts of the problem.

3.1.4 Interaction between Search and Simulation

Heuristic search requires strong evaluation functions to judge the plausibility of branches leading to complete hypotheses. However, in molecular biology many of the answers about plausibility of alternatives are not known *a priori*, but can only be known by experiment. Since laboratory experiments are typically very expensive, MOLGEN includes simulation models that can predict the time-course of an arbitrary experiment and thus can give some measure of the plausibility of that experimental procedure.

Several items mentioned above, including planning experiments, debugging and reasoning by analogy, will need to exploit the program's ability to simulate some aspects of experiments it is reasoning about. Complete simulations, of course, are also expensive, so we need to resolve issues about control of the simulation depending on the kinds of answers sought by the heuristic program.

3.1.5 Reformulating Available Methods

Each time our research group (9) has built another large AI program, we have learned more about how to do it better and faster next time. For example, the production rule interpreter in Heuristic DENDRAL (for special-purpose rules) became the general rule interpreter of MYCIN. One of the significant products of MOLGEN research will be the sets of ideas and programs for encoding and manipulating large amounts of knowledge about a scientific discipline. We have transferred some parts of the MOLGEN Units package to another project interested in building a knowledge base about AI methods and techniques. Making the tools used here available for use in new programs is an important aspect of our work, and is generally important for cumulation of knowledge in the AI field. In order to do this we must reformulate the methods so they are more generally applicable and more readily combined in diverse ways.

3.2 Significance to the Conduct of Experimental Science and to Science Policy

The exposition of a balanced view of the potentials of AI for practical applications in science faces many hazards. Enthusiasts make unlimited claims whose eventual realization is hard to disprove -- except that it is hard to say how long is 'eventual'. It would be difficult (though increasingly possible) to justify the investment in

---

(9) The MOLGEN project is part of a larger group known as the Heuristic Programming Project.

particular projects in terms of their utility for the object discipline; we believe, for example, that the DENDRAL project had to be assessed as a pathfinder, rather than for its specific utility to mass spectrometrists working today. The art has progressed to the point where MOLGEN may be expected to be at the margin -- that is to be the agency of concrete discoveries within a decade, advances that will compete in value with those achieved from comparable investments in existing tools in the biochemical laboratory. In suggesting new forms of working assistance, we do not imply that the creative imagination of scientists will be mimicked or displaced by AI programs over a broad domain of fact and insight: certainly not within our own immediate ambitions. However, we have intentionally chosen an experimental field, part of which is characterized by combinatorially elaborate contingency trees, some rigor of inference, and a fairly limited frame of relevant world-knowledge. These are precisely the conditions where programs can be expected to be of some help to human intellect, which thrives on the converse. A reexamination of the process of science may also be important to bolster and defend basic science at a policy level. The very justification for basic research is under critical, often even hostile scrutiny. Many quarters are asking such questions as "How much of the science progress of the past 30 years can be attributed to advances in knowledge connected with federally-supported research?" "Are our institutional arrangements and patterns of funding really the most appropriate for the most efficient 'transfer of technology' from the basic laboratory to useful applications?" Less often raised by external critics is, "To what extent does the present system support the most fundamental innovations within science itself; or does it inevitably focus overwhelming support on the most obvious, transparent questions and discourage more revolutionary kinds of inquiry?" Many of our colleagues reply to these attacks with well-intentioned promises and manifest good faith. Nevertheless, it is easy to show that many short-term advances have arisen from the most pragmatic kinds of investigation: empirical screening for antibiotics or antidiuretics has undoubtedly generated more life-saving therapeutic products than the most sophisticated molecular biology, up to the present moment. Indeed, salt-water, intelligently administered, has been one of the great life-savers of the recent era! It would be tragic to undermine the enormous long range potential of basic insight without a deeper analysis of the process by which knowledge and insight move from basic science into applied problems; and we just might find some ways to improve the system without wrecking it!

It would be premature to claim that computer programs *per se* will soon be delegated the major responsibility for "systematic identification of relevant knowledge", although they can already play a very helpful role in assisting human intelligence to correlate bibliographic data, and in other ways. However, the very process of implementing an "applied philosophy of science", which is the principal fore-work of developing a domain for the application of knowledge-based

AI, is exactly the kind of formal systematization needed for constructive efforts to facilitate technology transfer.

Although our substantive efforts are mostly concerned with the "micro-problems" of scientific inference, there may be more important treasures in a macro-perspective on the integration of scientific specialties. Improved systematization of scientific knowledge, should be an important side effect of these investigations in knowledge-engineering; and this may lead in turn to the recognition of remedial rents in the overall fabric. For example, it is dismaying to reflect on the delay of 35 years, from Beadle and Tatum's discovery of nutritional mutants in Neurospora, before similar ideas were applied to the biochemistry of human heart disease -- our most serious health problem by far. Is there no way to accelerate the benefits of such fundamental research? We will not get analytically persuasive or policywise sound determinations of such questions without more attention to the underlying process of scientific inquiry than unselfconscious scientists are wont to indulge.

These ideological implications of an 'applied philosophy of science' are complemented by some of the practical technologies of AI work. Our own research is greatly facilitated by access to the SUMEX-AIM system. This comprises not only a computational facility, but a national community of mutually interested investigators bound by effective computer-data communications. The development of formal representations of experimental science adds to the effectiveness with which the scientific community can enter into informed criticism of other's work, at the level of strategies of discovery and proof as well as in the exchange of laboratory data.

4 Budget

Budget to National Science Foundation
MOLGEN: A Computer Science Application to Molecular Genetics
Professors Edward A. Feigenbaum and Joshua Lederberg, Principal Investigators
June 1, 1978 - May 31, 1980

	6-1-78/5-31-79	6-1-69/5-31-80	Total Budget 6-1-78/5-31-80
Salaries and Wages			
Senior Personnel			
Edward A. Feigenbaum, Co-Principal Investigator 5% time Academic Year; 1 month Summer FTE: 1.45 months	5,093.	5,449.	10,542.
Joshua Lederberg, Co-Principal Investigator 5% time, Calendar Year; FTE: .45 months	∅	∅	∅
Bruce G. Buchanan, Adjunct Professor 25% time, Calendar Year; FTE: 3.00 months	8,684.	9,291.	17,975.
Other Staff			
Mark Stefik: Student Research Assistant 100% time, Summer Quarter; FTE: 3.00 months	2,862.		
Ph.D. Research Associate 75% time, effective 9-1-78 Year One FTE: 6.75 months	11,250.		
Year Two FTE: 9.00 months		15,844.	29,956.
Peter Friedland: Student Research Assistant 100% time, Summer Quarter; FTE: 3.00 months	2,862.		
Ph.D. Research Associate 100% time, effective 9-1-78 Year One FTE: 9.00 months	15,000.		
Year Two FTE: 12.00 months		21,125.	38,987.
Student Research Assistant, Computer Science, to be named 50% time, Academic Year 100% time, Summer Quarter FTE: 7.50 months	7,387.	7,838.	15,225.
Post-Doctoral Scholar, Genetics, to be named 100% time, Calendar Year FTE: 12.00 months	14,630.	15,508.	30,138.
Secretarial Assistance 20% time, Calendar Year FTE: 2.40 months	2,095.	2,245.	4,340.
Subtotal, Salaries	$ 69,863.	$ 77,300.	$ 147,163.

	6-1-78/5-31-79	6-1-79/5-31-80	Total Budget 6-1-78/5-31-80
Staff Benefits 19.0% 9-1-77/8-31-78 20.3% 9-1-78/8-31-79 21.6% 9-1-79/8-31-80	13,952.	16,442.	30,394.
Permanent Equipment Computer Terminal, Datamedia, including modem	2,835.	-	2,835.
Expendable Supplies and Equipment	875.	1,000.	1,875.
Travel Domestic: professional meetings and trips to collaborate in New Mexico	1,000.	1,000.	2,000.
Publications Costs	700.	800.	1,500.
Computer Costs - All services provided by SUMEX computer facility	∅	∅	∅
Other Costs Communications (telephones)	750.	900.	1,650.
Subtotal, Direct Costs	89,975.	97,442.	187,417.
Indirect Costs, Excluding Permanent Equipment - 58%	50,542.	56,517.	107,059.
Total Budget	$ 140,517.	$ 153,959.	$ 294,476.

BUDGET NOTES

1. Staff salaries

Mark Stefik and Peter Friedland, currently Computer Science Ph.D. thesis students, have been working with the MOLGEN project since its inception. They have invested very heavily in personal time and effort, and the MOLGEN project has invested resources, to bring them to the point of being highly informed "bridge" scientists between Computer Science and Genetics. A high return on this investment, from both the scientific and economic viewpoints, will be obtained by retaining these young scientists for two more years as post-doctoral researchers. Each is expected to complete his Ph.D. thesis in August,1978, and will therefore be paid at pre-doctoral salary rates for the summer of 1978. Thereafter, or upon completion of the Ph.D., they will be paid at "new Ph.D." salaries, i.e. approximately what a new assistant professor would be paid (annualized). Stefik is budgeted at 75% effort under the assumption that other support can be found for the remaining 25% of his time. Thus, he will be on the project 100%, but paid only 75% from this budget.

2. Permanent equipment

We are requesting one more terminal of the "standard" type that is used for 1200 baud access to SUMEX, i.e. the Datamedia. MOLGEN is a compute-intensive project. In the continuation period, we anticipate that the chief bottleneck to effective computer use will not be computer access or cycles but terminal access. The SUMEX facility does not supply terminals to individual projects, but expects each project to supply its own needs. MOLGEN has two terminals now, but one more will be needed as the project expands in effort, scope, and size. This need is modest considering the fact that all other computer support is supplied to the project at no charge.

3. Phone charges

These include not only charges for ordinary project business (small part) but also charges for phone line communication to the computer (large part).

4. Travel

Funds for domestic travel are needed to support travel to occasional professional conferences in Computer Science, particularly the annual conference of the SUMEX-AIM community (the conference supports some of this; individual projects pick up the remainder); and to support travel to collaborate with Professor Martin and her group at the University of New Mexico.

5 Resources

Professors Edward Feigenbaum, Bruce Buchanan and Nancy Martin (New Mexico) will direct the computer science research of the MOLGEN continuation project. The principal project staff members at Stanford will be Peter Friedland and Mark Stefik. Both will complete their Ph.D. theses on MOLGEN early in the continuation period, and both have agreed to stay on for the MOLGEN continuation. An additional computer science student will be taken on, hopefully leading to another MOLGEN project Ph.D. thesis.

Molecular genetics knowledge, expertise, insights, techniques, and experimental heuristics will be provided by the researchers in Professor Joshua Lederberg's laboratory at Stanford, including graduate student Jerry Feitelson and a post-doctoral research fellow to replace Dr. S. D. Ehrlich. Professor Lederberg himself will provide substantial amounts of time on a regular basis for directing the project from the genetics viewpoint.

Offices for the MOLGEN project will be provided within the Stanford Heuristic Programming Project so as to foster interaction and exchange of ideas with workers on similar projects. Active projects within the Heuristic Programming Project include:

1) DENDRAL, a knowledge-based system for the analysis of organic compounds from spectrometric data.

2) META-DENDRAL, a system for inducing rules of mass spectrometry and n.m.r. spectroscopy from instrument data.

3) MYCIN, a system for the diagnosis and treatment of infectious disease. Approximately thirty workers including faculty, research associates, and graduate students are involved among the projects. All of these projects are active in the design of intelligent systems for specific domains of science and medicine providing sources of problems and insight concerning complex reasoning processes. There has been considerable synergy among the various projects.

4) PUFF, a MYCIN-like system for diagnosis of pulmonary function disorders.

5) CRYSALIS, a system for inferring the structure of proteins from electron density maps derived from x-ray crystallographic data.

6) VM, a heuristic process control system for assisting physicians in the management of a breathing-assistance machine in the intensive care units of hospitals.

7) AGE, an attempt to build: a software package of AI techniques and methods for problem solving and hypothesis formation; and its associated user-interface.

The superb computing facilities of the NIH-supported SUMEX-AIM timesharing facility will be available at no charge to this project. The SUMEX-AIM facility, with Prof. Lederberg as principal investigator, is a national resource for the application of artificial intelligence techniques to problems in biology and medicine. Resources to be provided will include all CPU-time and storage required. Those involved at Stanford will be operating through hard-wired or dial-up equipment to the SUMEX PDP-10, while those at the University of New Mexico will access the system through either the ARPA network or TYMNET.

The SUMEX-AIM facility is a powerful interactive computing system open to a national community. Interlisp and other high level languages are available and supported by a large system staff. Many convenient text editors for developing programs are provided. The TENEX operating system supports flexible file handling and sophisticated storage management for a highly interactive computing environment.

Appendix I

GLOSSARY

AGE — Stanford project ("Attempt to GEneralize") to build a general package of AI methods.

AI — Artificial Intelligence

AI "toolbox" — A set of AI concepts embodied in programs that are general enough to be used to construct problem solving programs in many different domains.

alkaline phosphatase — An enzyme that removes terminal phosphates from nucleic acids, and whose optimum pH is alkaline.

AT-rich DNA — DNA containing a high proportion of Adenine + Thymine base pairs. Because of the Watson-Crick base pairing rules, A/T=1 and G/C=1, but (A+T)/(G+C) ratios can can vary widely between DNA of different species and even different regions of the same DNA molecule.

B. subtilis — A common soil bacterium often used in genetic experiments.

bacteriophage — Viruses that multiply in bacteria.

base sequence — A string of nucleotides in a nucleic acid.

bottom-up process — A program that puts together inferences from data without the benefit of global expectations and goals.

CONGEN — Constrained Generator of molecular structures for DENDRAL.

dalton — A unit of mass equal to that of a single hydrogen atom.

data-driven procedure — bottom-up process

demon — a procedure in a program that is triggered by an event, as opposed to being executed in the "normal" execution of a sequential program.

denaturation — The loss of the native configuration of a macromolecule resulting, for example, from heat treatment, extreme pH changes, chemical treatment, or other denaturing agents. It is usually accompanied by the separation of strands (in DNA) and the loss of biological activity.

DENDRAL — Heuristic program for generating and testing organic molecular structures/as candidate explanations of empirical data.

digestion — With reference to enzymes, implies the cleaving of chemical bonds in the target molecule. For example, exonucleases "erode" or remove terminal nucleotides, restriction enzymes cut at the internal recognition sequences.

dimer — A concatenated DNA structure consisting of two identical constituents.

discrimination experiment — A series of experimental steps designed to conclude whether structures are identical or not.

DNA — Deoxyribonucleic acid. A polymer of deoxyribonucleotides (see nucleotide definition). Can exist as double or single strands. The genetic material of all cells and the central molecule in molecular genetics.

domain-specific critic — A procedure which applies specific genetics knowledge to problem solving, as opposed to general problem-solving knowledge.

EcoRl — A restriction enzyme isolated from a strain of E. coli that cleaves DNA at site-specific regions along the molecule. Its recognition site is 5'-GAATTC-3'.

EDNA — The DNA-structure-editor for MOLGEN.

electron microscopy — Abbreviated EM. A high-resolution technique for visualizing material that uses beams of electrons instead of light rays. Resolutions of about 10^-7 cm are posible with biological materials.

electrophoresis — An experimental technique used to separate, purify, and measure the molecular weight

of molecules having an electric charge in solution.

endonuclease — An enzyme that cuts DNA backbone chains internally.

enzyme — Protein molecule capable of catalyzing a specific chemical reaction.

E. coli — A common intestinal bacterium: the most intensively studied organism except for man.

event-driven procedure — demon

exonuclease — An enzyme that digests DNA from the ends of strands.

experiment planning — An activity characterized by the production of a sequence of experimental steps to achieve a goal.

focus rules — Focus of attention procedures. Items of knowledge that guide a program to the most relevant parts of the problem or the most useful subroutines.

gaps — An internal feature of double-stranded DNA which is a region of unpaired nucleotides due to the excision of a string on one strand.

HEARSAY — AI program written at Carnegie-Mellon University to understand spoken English. Integrates inferences made by multiple experts.

hierarchical planning — AI techniques refined by Sacerdoti which uses a hierarchy of descriptions to plan an efficient problem solution procedure.

inspector — Domain-specific critic

INTERLISP — A powerful extension of the LISP programming language.

KRL-0 — A programming language (knowledge representation language) developed at Stanford and Xerox, Palo Alto Research Center.

ligase — An enzyme capable of covalently joining parts of, or entire DNA molecules together.

ligation — The enzymatic joining together of DNA molecules.

linear DNA — Double stranded DNA that is not covalently closed at its termini.

meta-rules — Rules for a program that mentions domain-specific rules, i.e, to prune or reorder the set of rules relevant for problem solving in specific contexts.

molecular adapter — A chemically synthesized segment of DNA that is utilized to join together DNA molecules which do not have complementary termini for ligation.

MOLGEN — Computer program for reasoning in molecular genetics. Main subject of this proposal.

monomer — A single DNA molecule (or nucleotide) that has not undergone polymerization (viz. a unit character capable of assembly into a string).

MYCIN — Medical diagnosis and therapy recommendation program developed at Stanford.

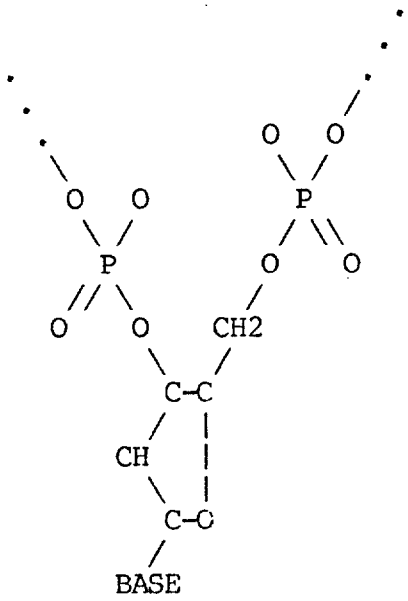
nicks — A local interruption in the phosphodiester backbone of DNA. No genetic information is missing due to this structural anomaly.

nuclease — An enzyme which breaks chemical bonds in the DNA phosphodiester backbone. Consists of endonucleases and exonucleases.

nucleotide — The building blocks of DNA consisting of a purine (Adenine or Guanine) or a pyrimidine (Thymine or Cytosine) linked to a deoxyribose sugar with a phosphate group also linked to adjacent sugar. Adjacent nucleotides are linked together through a phosphate group and a hydroxyl group on the sugar component (see phosphodiester).

pH — The negative logarithm of the effective hydrogen ion concentration or hydrogen ion activity in gram equivalents per liter. Used in expressing both acidity and alkalinity on a scale whose values run from 0 to 14; 7 representing neutrality, less than 7 increasing acidity, >7 increasing alkalinity. DNA exists in native form between pH values of 5 and 12.

phosphodiester — The chemical link between adjacent nucleotides. The following diagram of its structure was drawn using CONGEN [6]:



plan schema — A sketch of a procedure describing the plan for an experiment in abstract, general terms.

planning islands — Partial solutions to a problem found by a planning program. "Stepping stones" to a complete solution.

planning rules — Procedures or items of knowledge that aid a program in constructing a problem solving plan.

plasmid — Extrachromosomal DNA molecules which are double stranded, circular, and supercoiled. They range in size from about 5*10^6 daltons to near 10^8. Small plasmids can exist in many (more than 50) copies per cell while large ones are maintained at one or two. They are often used as vectors for amplifying and transferring DNA from one organism to another.

poly-A region (sequence) — A homopolymeric sequence of adenine nucleotides. Implies a poly-T region on the complementary strand.

poly-C — Homopolymeric cytidine nucleotides.

poly-G — Homopolymeric guanine mucleotides.

poly-T — Homopolymeric thymine nucleoties.

polymerase — Enzymes that are catalysts for nucleic acid chain growth.

pre-conditions — Premise clauses of conditional sentences that must be satisfied before the consequent actions are taken.

production rules — Conditional sentences used to encode inferential knowledge for a program.

prototype — The type of unit created for representing information about general concepts. Features are defined by slots associated wih the prototype.

restriction enzyme — Site-specific endonucleases used frequently in molecular genetic manipulations. Allow previously impossible experiments to be performed due to their ability to cleave DNA at reproducible locations allowing rearrangements within and between molecules.

RNA — Ribonucleic acid. Typically single stranded, is a polymer of ribonucleotides connected by phosphodiester bonds.

schema/rule schema/program schema — An abstract, generalized representation of a concept or program. In MOLGEN, program schemata (or rule schemata) are represented as Units with slots defined for important features.

SECS — Chemical synthesis planning program developed by Prof. Todd Wipke (U.C. Santa Cruz).

self-circularization — Ligation of the ends of the same DNA resulting in a circular, covalently closed molecule.

self-ligation — Ligation of a DNA molecule to itself, resulting in in a circular molecule. Catalyzed by ligase.

sequencing experiment — A technique to determine the order of nucleotides in a strand of DNA.

slots — Pre-defined features of objects for which values are sought.

SMALLTALK — Display-oriented programming language developed at Xerox, Palo Alto Research Center.

sticky ends — A condition of partial single-strandedness at the termini of DNA molecules, allowing base pairing in that region. Restriction enzymes often leave sticky ends, greatly facilitating the rearrangements of DNA.

STRIPS — Robot plnning program developed at SRI.

SUMEX-AIM — NIH-sponsored computer resource for applications of artificial intelligence in medicine.

Teiresias — AI program that acquires inference rules for MYCIN and guides MYCIN reasoning.

TENEX — Operating system for the DEC KI-10 system running at the SUMEX-AIM facility.

Thy gene — A gene coding for the enzyme, thymidylate synthetase. This enzyme is crucial in enabling a bacterium possessing it to produce thymidine, a constituent of DNA.

top-down process — A program that works from general principles, testing data against expectations and goals, often working by dividing complex problems into simpler ones.

Units — Basic element of representation in MOLGEN. Units are organized in a hierarchy to facilitate the representation of class-subclass and prototype-instance relationships. Units are used for representing processes as well as concepts.

vector — A self-propagating DNA molecule that can be used to link DNA sequences of interest. Vectors can be one of several replicating plasmid or bacteriophage DNAs.

world states — Representation of the state of an experiment at any given time. The "world" for the program is the limited set of objects and operations relevant to a specific experiment.

3'-end/ 5'-end — Related to the direction of the phosphodiester bonds in the backbone of DNA molecules. Each strand thus has one 3' end and one 5' end.

Appendix II

EDNA -- The Editor for DNA

The following is an actual session with the MOLGEN knowledge acquisition system recorded from SUMEX. Comments preceded by a semicolon have been inserted to clarify some aspects of the dialog.

```
@ue                                        ;UE is the name of the Unit
                                           ;Editor. Here it is being called
(Version  4-OCT-77 08:56:16)               ;from TENEX

Welcome to the MOLGEN Unit Editor.  Type ? anytime for assistance.
The symbol : indicates that the editor is waiting for your input.
Two characters are enough for command recognition.  You may type ahead
responses for a command.

Name of Network: jerry                     ;Jerry is the name of an
                                           ;existing Knowledge Base on
                                           ;file.
:create testl root specialization          ;A new unit TEST1 is created.
Give a value for the DESCR slot            ;UE asks for documentation.
Text Editor
te: Test unit to demonstrate the DNA structure editor
te: done                                   ;User indicates he is done
                                           ;with documentation.
Do you want to see what slots have already been filled? yes

DESCR:          (U)   from ROOT     <DESCR>
Test unit to demonstrate the DNA structure editor
MODIFIER:       (U)   from ROOT     <MODIFIER>   STEFIK
CREATOR:        (U)   from ROOT     <CREATOR>    STEFIK
MODIFIED:       (U)   from ROOT     <MODIFIED>    6-OCT-77 10:22:03
CREATED:        (U)   from ROOT     <CREATED>     6-OCT-77 10:22:03

                                           ;Note that the system has
                                           ;automatically recorded the
                                           ;author, date and time of the
                                           ;new unit.

You can now create new slots or edit old ones.  When through type DONE
EDIT: create substrate
Datatype: dna                              ;DNA is a datatype.
Role: r                                    ;"Role" controls transmission
                                           ;of the value in the
                                           ;substrate slot if we make
```

```
                                        ;specializations of TEST1.
Is it a dynamic slot? n
DNA Editor                              ;Since the datatype is DNA,
                                        ;we get the DNA editor.
Copy or Create anew? create
Segment Type: ?                         ;A "?" may always be typed to
                                        ;tell the system to clarify
                                        ;what it expects for a
                                        ;response.

Choose one of the following Segment Types.

Type    Description
LE      Length Segment.  Indicates number of nucleotides in a region.
BA      Base Segment.  Indicates actual Base Sequence.
SI      Cut Site for enzyme.
Segment Type: length
Length: ?                               ;Another "?"

Indicate number of nucleotides as in the following examples:

You Type        Meaning

5               5 nucleotides
100 200         Between 100 and 200 nucleotides.
1K  1.3K        Between 1000 and 1300 nucleotides. (K=1000) No Spaces!
R 1K 1.3K       Same as above except RNA instead of DNA.
Length: 2.5k
                                        ;The initial segment is now
                                        ;specified.  We may now issue
                                        ;any legal EDNA command.
edna: print

DNA Printer (Version 19-SEP-77 )
     1
     (2500)
     ------

edna: insert 1 3' bases attacg
edna: print

     1      2 3 4 5 6 7                 ;EDNA presents its structures
     (2500) A T T A C G                 ;pictorially.  Segments are
     ------------------                 ;referenced by number.

edna: mirror 4 to 7                     ;"Mirror" means to add a
edna: print                             ;parallel DNA strand.

     1      2 3 4  5  6 7
     (2500) A T T  A  C G               ;EDNA knows about
```

```
     ------------------                    ;complementary bases.
                ---------
               A  T  G C
               11 10 9 8

edna: break 4 h                           ;Break a bond.  A Break
edna: print                               ;command can specify 5', 3',
                                          ;or H bonds.
     1      2 3 4 5  6 7
     (2500) A T T A  C G
     ------------------
                /------                   ;Segment 11 is depressed
               -- T  G C                  ;to indicate the broken bond.
               A  10 9 8
               11

edna: connect 7 3' 8
edna: print

     1      2 3 4 5  6 7
     (2500) A T T A  C G->8               ;A similar notation is used
     -------------------|                 ;to indicate Hairpin loops.
                /-------|
               -- T  G C
               A  10 9 8
               11

edna: undo                                ;EDNA can "undo" any of its
                                          ;structure changing commands
CONNECT undone.
edna: done                                ;User is finished editing
                                          ;this structure.  He returns to
                                          ;the slot editor.

EDIT: print all
DESCR:          (U)   from ROOT      <DESCR>
Test unit to demonstrate the DNA structure editor
MODIFIER:       (U)   from ROOT      <MODIFIER>  STEFIK
CREATOR:        (U)   from ROOT      <CREATOR>   STEFIK
MODIFIED:       (U)   from ROOT      <MODIFIED>   6-OCT-77 10:22:09
CREATED:        (U)   from ROOT      <CREATED>    6-OCT-77 10:22:03
SUBSTRATE:      (R)   *Top*          <DNA>       (Renumbering 11)


     1      2 3 4 5  6 7
     (2500) A T T A  C G
     ------------------
                /------
               -- T  G C
               A  10 9 8
               11
```

```
EDIT: done                          ;User is done with this unit.
:done                               ;User is done with this
                                    ;knowledge base.
Save JERRY? no                      ;He doesn't save his changes
Bye                                 ;because this was just a
                                    ;demonstration.
```

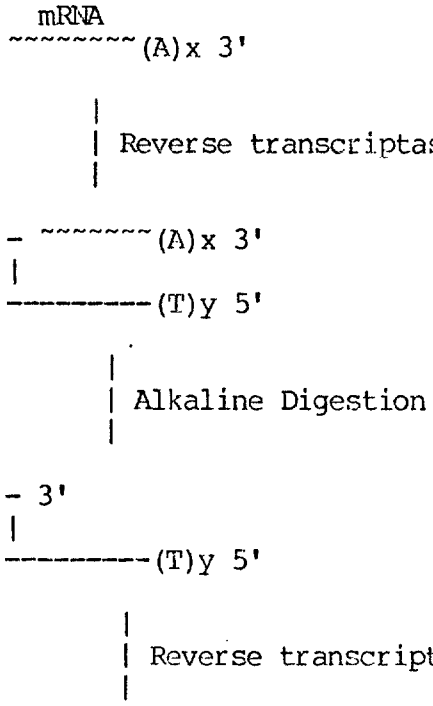
October 27, 1977

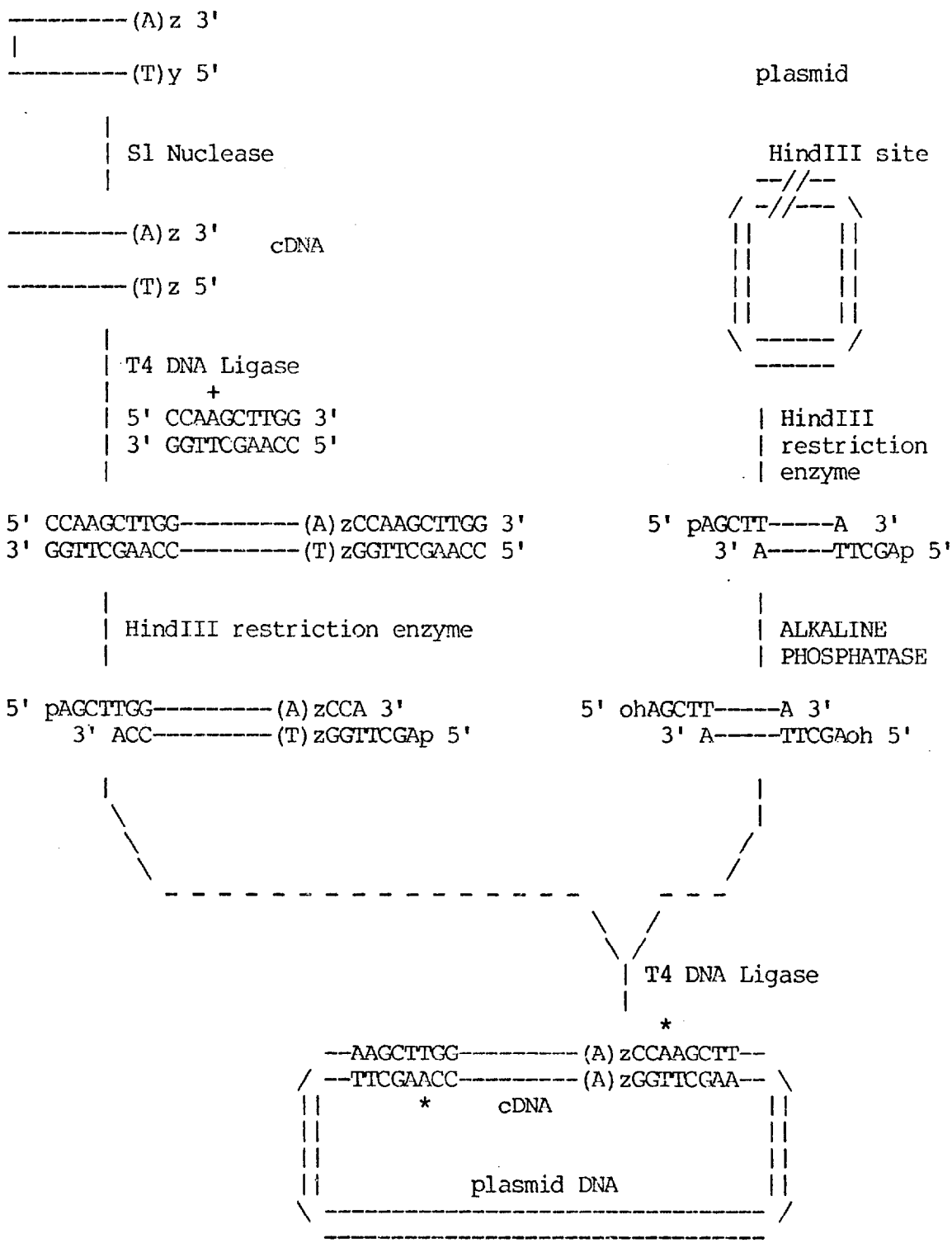
Appendix III

A Genetic Planning Example

This section is intended to extend the range of genetic examples for which MOLGEN is envisioned being applied. In particular, the recent cloning of the rat insulin gene in *E. coli* [38] has been achieved using a simple additional step to the usual experimental protocol. It is asserted that the genesis of this efficiency-improving step can be found in the relatively simple application of knowledge about enzyme properties and DNA ligation kinetics.

The basic experimental outline is seen below, modified from [38]. It closely follows the 'classical' recombinant DNA methodology, with a few additional steps. The one we wish to focus on most closely in this discussion is the application of Alkaline Phosphatase to the plasmid vector after cutting of the plasmid by the restriction enzyme HindIII.




---------(A)z 3'
|
---------(T)y 5'
plasmid
| S1 Nuclease
|
HindIII site
---------(A)z 3'
cDNA
---------(T)z 5'
| T4 DNA Ligase
| +
| 5' CCAAGCTTGG 3'
| 3' GGTTCGAACC 5'
| HindIII
| restriction
| enzyme
5' CCAAGCTTGG---------(A)zCCAAGCTTGG 3'
3' GGTTCGAACC---------(T)zGGTTCGAACC 5'
5' pAGCTT-----A 3'
3' A-----TTCGAp 5'
| HindIII restriction enzyme
| ALKALINE
| PHOSPHATASE
5' pAGCTTGG---------(A)zCCA 3'
3' ACC---------(T)zGGTTCGAp 5'
5' ohAGCTT-----A 3'
3' A-----TTCGAoh 5'
| T4 DNA Ligase
*
--AAGCTTGG---------(A)zCCAAGCTT--
--TTCGAACC---------(A)zGGTTCGAA--
* cDNA
plasmid DNA

The following steps were carried out. First, insulin messenger RNA was purified from B cells in the rat pancreas. This was reverse transcribed into a hybrid DNA/RNA structure by the use of avian myeloblastosis virus (AMV) reverse transcriptase and the RNA selectively degraded by raising the pH. A double-stranded DNA form was synthesized by incubating this with deoxynucleoside triphosphates and the AMV reverse transcriptase (a DNA polymerase could have been used). The hairpin at the end of the molecules and any non-base paired regions were removed with the single-strand specific nuclease S1.

The resulting molecular structure is termed cDNA, or copy DNA, because it should contain the precise genetic information contained in the gene coding for the insulin messenger RNA. This is the in-vitro synthesized segment that is to be cloned in bacterial recipients for amplification and analysis.

A recently developed technique for ligating chemically synthesized restriction site linkers (adapters) [35] to cDNA was used in order to produce cDNA molecules with cohesive termini after digestion with a restriction endonuclease enzyme. Ligating the resulting cDNA to plasmid DNA cut with the identical restriction enzyme would create a recombinant plasmid which could then be cloned in a suitable bacterial host. Specifically, a decamer linker containing a site for HindIII was covalently joined to the ends of the cDNA with T4 DNA ligase, and then cleaved with HindIII; pMB9, a 3.5 million dalton plasmid conferring tetracycline resistance with a single site for HindIII, was also cut with the same endonuclease.

The usual procedure would be to now straightforwardly ligate these two molecules together creating the desired recombinant molecule. Kinetic theory [11] suggests that in order to insure ligation of most of the cDNA molecules to plasmid DNA, it is necessary to add a molar excess of plasmid DNA. However this would result in the majority of the plasmids simply self-circularizing without an insert of cDNA, and thus most the transformed cells would contain only pMB9 and not the desired recombinant plasmids. Here is where the novel step of removing terminal phosphates on the plasmid was generated.

Several sources of knowledge need be brought to bear in order to understand the basis of this new optimization step. Firstly, we need to know that alkaline phosphatase removes 5' terminal phosphates from the HindIII endonuclease-generated ends of the plasmid. Secondly, knowledge about the requirement of the T4 ligase for a phosphate end configuration allows us to infer that removing the phosphate ends prevents self-ligation of the plasmid DNA.

Thirdly, the kinetic theory of ligation [11] combined with a rule that says, "In a process that involves two or more competing components, you can optimize one process by inhibiting the other(s)",

should tell us that circle formation is now dependent on the insertion of a DNA fragment containing 5'-phosphorylated termini: the cDNA. Finally, since transformation is directly linked to the DNA source, the one step inference is: "Only recombinant plasmids will transform the recipient bacteria".

A side effect that needs to be dealt with is the fact that the recombinant plasmids generated after phosphatase and ligase treatments will have two nicks, represented as asterisks in the figure, and that this has no known effect on transformation efficiency.

The application of alkaline phosphatase to remove terminal phosphates from a restriction enzyme-cleaved vector (e.g. plasmid) to eliminate self-ligation is a novelty that "should" have been obvious to any investigator working in this field. In fact, three or four years passed before Ullrich et. al. utilized it. One can only speculate as to the reasons why. However two related responses arise in this context. First, there are a very large number of DNA reagents available to the investigator (enzymes, chemical and separative techniques) so the number of possible combinations are vast. Secondly, especially with a well focused goal such as the ligation optimization step discussed above, people tend to think along relatively stereotyped paths, e.g. previously developed protocols. A computer system, such as MOLGEN, with a complex knowledge base and a good set of heuristic rules, will be likely to uncover novel applications of well known tools, precisely along the lines of the example just presented.

should tell us that circle formation is now dependent on the insertion of a DNA fragment containing 5'-phosphorylated termini: the cDNA. Finally, since transformation is directly linked to the DNA source, the one step inference is: "Only recombinant plasmids will transform the recipient bacteria".

A side effect that needs to be dealt with is the fact that the recombinant plasmids generated after phosphatase and ligase treatments will have two nicks, represented as asterisks in the figure, and that this has no known effect on transformation efficiency.

The application of alkaline phosphatase to remove terminal phosphates from a restriction enzyme-cleaved vector (e.g. plasmid) to eliminate self-ligation is a novelty that "should" have been obvious to any investigator working in this field. In fact, three or four years passed before Ullrich et. al. utilized it. One can only speculate as to the reasons why. However two related responses arise in this context. First, there are a very large number of DNA reagents available to the investigator (enzymes, chemical and separative techniques) so the number of possible combinations are vast. Secondly, especially with a well focused goal such as the ligation optimization step discussed above, people tend to think along relatively stereotyped paths, e.g. previously developed protocols. A computer system, such as MOLGEN, with a complex knowledge base and a good set of heuristic rules, will be likely to uncover novel applications of well known tools, precisely along the lines of the example just presented.

References

1. Bobrow D.G. and Winograd T., Experience with KRL-0: One Cycle of a Knowledge Representation Language, 5IJCAI:213-222 (1977)

2. Bobrow D.G. and Winograd T., An Overview of KRL, a Knowledge Representation Language, Cognitive Science 1 (1977)

3. Brachman R.J., What's in a Concept: Structural Foundations for Semantic Networks, BBN Report No. 3433 (1976)

4. Brown R., Use of Analogy To Achieve New Expertise, Artificial Intelligence Laboratory, MIT, AI-TR-403 (1977)

5. Buchanan, B.G. and Dennis Smith, "Computer Assisted Chemical Reasoning", in Proceedings of the III International Conference on Computers in Chemical Research, Education and Technology, Plenum Publishing, pp. 388-408 (1977)

6. Carhart, R.E., Smith, D., Brown, H., and Djeralls, C., Applications of artificial intelligence for chemical inference. XVII. An approach to computer-assisted elucidation of molecular structure J. Am. Chem. Soc. 97:5755-5762 (1975).

7. Davis R., Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases, Stanford Computer Science Department Report No. STAN-CS-76-552 (1977)

8. Davis R., Generalized Procedure Calling and Content-Directed Invocation, SIGPLAN Notices, 12:45-54 (1977)

9. Dershowitz N. and Manna Z., The Evolution of Programs: A System for Automatic Program Modification, Stanford University Artificial Intelligence Laboratory (1976)

10. Dijkstra E.W., Notes on Structured Programming, in Dahl,O., Dijkstra, E.W. and Hoare, C.A.R. in Structured Programming, Academic Press (1972)

11. Dugaiczyk, A., Boyer, H. and Goodman, H., Ligation of EcoRl Endonuclease- generated DNA Fragments into Linear and Circular Structures, J. Mol. Bio. 96:171-184 (1975)

12. Ehrlich S.D., Bursztyn-Pettegrew H., Stroynowski I. and Lederberg J., Expression of the thymidylate synthetase gene of the Bacillus subtilis bacteriophage Phi-3-T in Escherichia coli, Proceedings of the National Academy of Sciences USA, 73:4145-4149 (1976)

13. Evans T.G., A Program for the Solution of Geometric Analogy Intelligence Test Questions, in First Order Mathematical Logic, Margaris, A., Blaisdell Publishing Company, Waltham, Mass. (1967)

14. Feitelson J.S. and Stefik M.J., A Case Study of the Reasoning in a Genetics Experiment, Heuristic Programming Project Working Paper 77-18, Computer Science Department, Stanford University (1977)

15. Fikes R.E. and Nilsson N.J., Learning and Executing Generalized Robot Plans, Artificial Intelligence, 3:251-288 (1972)

16. Fikes R.E. and Nilsson N.J., STRIPS: A New Approach to Applications of Theorem Proving to Problem Solving, Artificial Intelligence 2:189-208 (1971)

17. Feigenbaum E.A., The Art of Artificial Intelligence: I. Themes and Case Studies of Knowledge Engineering, 5IJCAI:1014-1029 (1977)

18. Gerhart S.L., What Goes Down Should Also Come Up: Some Issues about Abstraction, 5th Texas Conference on Computing Systems (1976)

19. Gerhart S. L., Knowledge About Programs: A Model and Case Study, Proc. of Intl. Conf. on Reliable Software, Los Angeles (1975)

20. Goldberg A. and Kay A., SMALLTALK-72 Instruction Manual, Xerox Corporation (1976)

21. Goldstein I.P. and Roberts R.B., NUDGE, A Knowledge-based Scheduling Program, 5IJCAI:257-263 (1977)

22. Hayes-Roth F. and Lesser V.R., Focus of Attention in the Hearsay-II Speech Understanding System, 5IJCAI:27-35 (1977)

23. Kling R.E., Reasoning By Analogy With Applications To Heuristic Problem Solving: A Case Study, Stanford Computer Science Department Report No. CS-216 (1971)

24. Martin N., Friedland P., King J. and Stefik M., Knowledge Base Management for Experiment Planning in Molecular Genetics, 5IJCAI:882-887 (1977)

25. McDermott D., Vocabularies for Problem Solver State Descriptions, 5IJCAI:229-234 (1977)

26. Miller M.L. and Goldstein I.P., Structured Planning and Debugging, 5IJCAI:773-779 (1977)

27. Minsky M.A., A Framework for Representing Knowledge, in Winston P (ed) The Psychology of Computer Vision, New York: McGraw-Hill (1975)

28. Notani, N.K. and Setlow, J.K., Mechanism of Bacterial Tranformation and Transduction, in Progress in Nucleic Acid Research and Molecular Biology 14:39-100 (1974)

29. Platt J.R., Strong Inference, Science 146:347 (1964)

30. Rieger C. and Grinberg M., The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms, 5IJCAI:250-256 (1977)

31. Roberts, R.J. CRC Critical Reviews of Biochemistry 4 :123-164 (1976)

32. Sacerdoti E.D., The Nonlinear Nature of Plans, 4IJCAI,:206-214 (1975)

33. Sacerdoti E.D., Planning in a Hierachy of Abstraction Spaces, 3IJCAI:412-422 (1973)

34. Schaffner K., Logic of Discovery and Justification in Regulatory Genetics, Stud. Hist. Phil. Sci. 4:349-385 (1974)

35. Scheller R.H., Dickerson R.E., Boyer H.W., Riggs A.D. and Itakura K., Chemical Synthesis of Restriction Enzyme Recognition Sites Useful for Cloning, Science 196:177-180 (1977)

36. Stefik M.J. and Martin N., A Review of Knowledge Based Problem Solving as A Basis for a Genetics Experiment Designing System, Stanford Computer Science Department Report No. STAN-CS-77-596 (1977)

37. Sussman G.J., The Virtuous Nature of Bugs, Proceedings of the AISB Summer Conference (1974)

38. Ullrich A., Shine J., Chirgwin J., Pictet R., Tischer E., Rutter W.J., and Goodman H.M., Rat Insulin Genes: Construction of Plasmids Containing the Coding Sequences, Science 196:1313-1319 (1977)

39. Walker D.E., Paxton W.H., Grosz B.J., Hendrix G.G., Robinson A.E., Robinson J.J., Slocum J., Procedures for Integrating Knowledge in a Speech Understanding System, 5IJCAI:36-42 (1977)

40. Wipke W.T., Computer-Assisted Three Dimensional Synthetic Analysis, in Computer Representation and Manipulation of Chemical Information, Wipke W.T. et.al. (eds.) John Wiley (1974)

41. Woods B., What's in a Link?, in Bobrow and Collins (eds.) Representation and Understanding (1975)