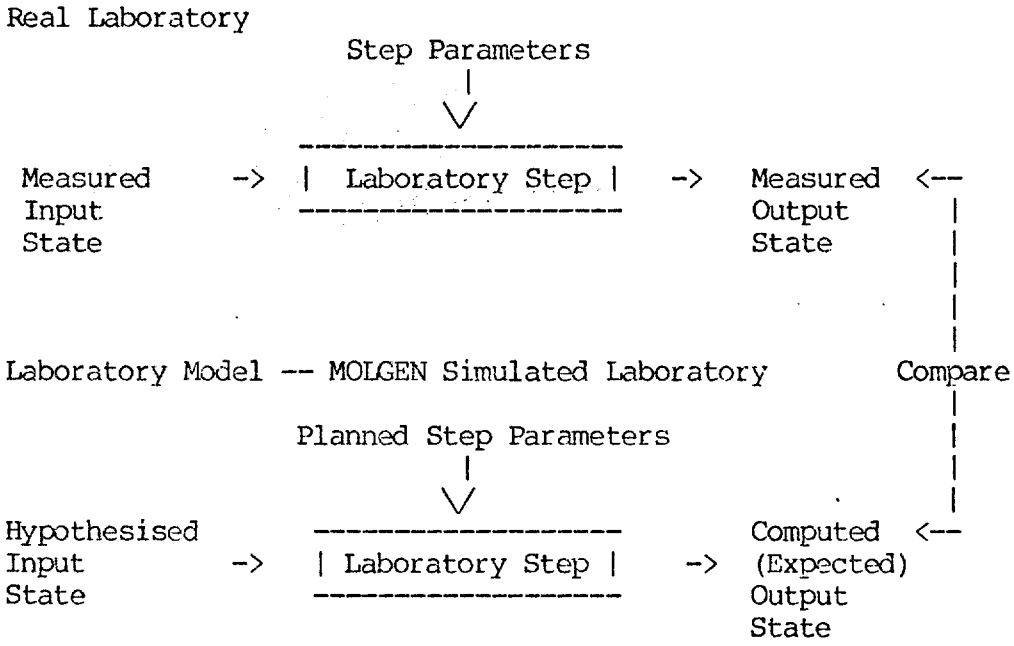


In this diagram, we assume that a complete plan for a laboratory experiment has been developed. In this diagram, we will focus on a single step in the experiment. The upper part of the diagram corresponds to the step as carried out in the laboratory -- illustrating that a laboratory sample undergoes 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.

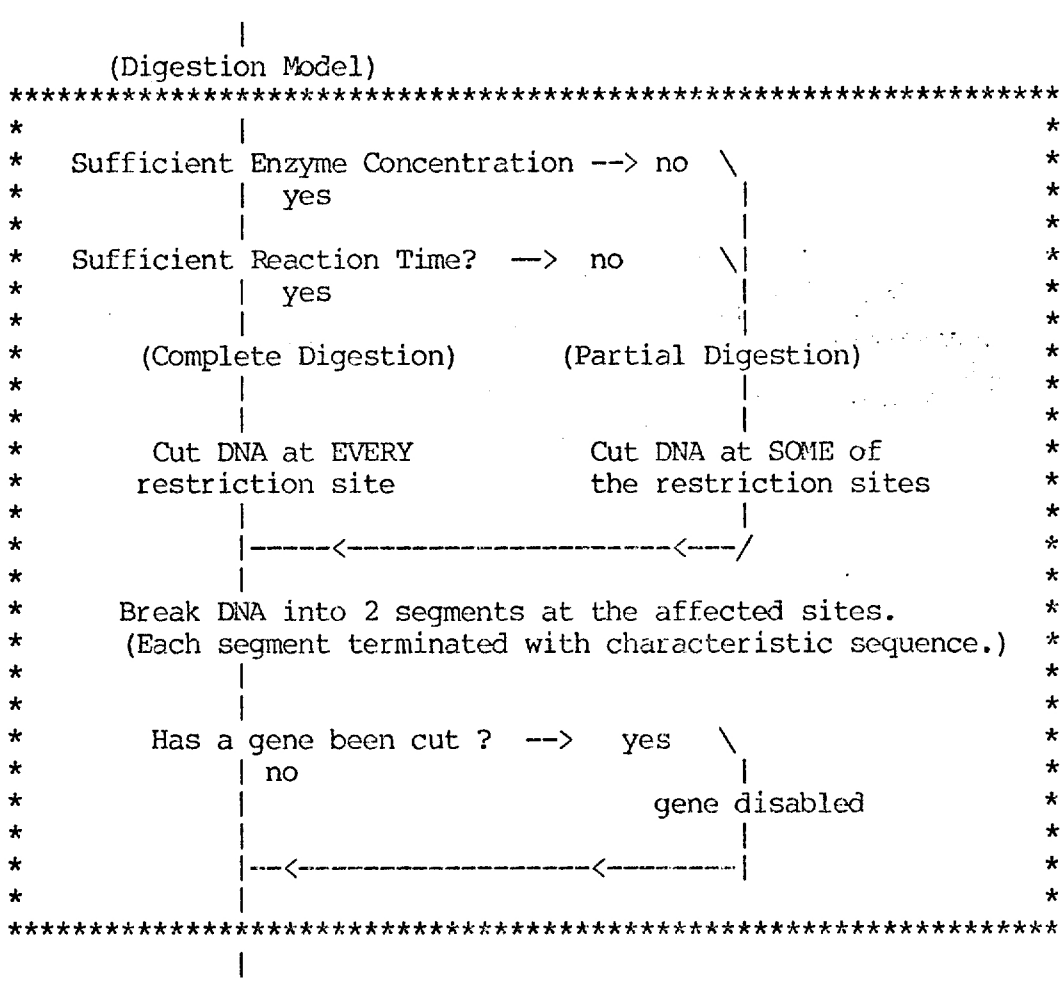
Both of these hypotheses may be derived by analysis of the process models for digestion (7) and transformation. These models are given below.

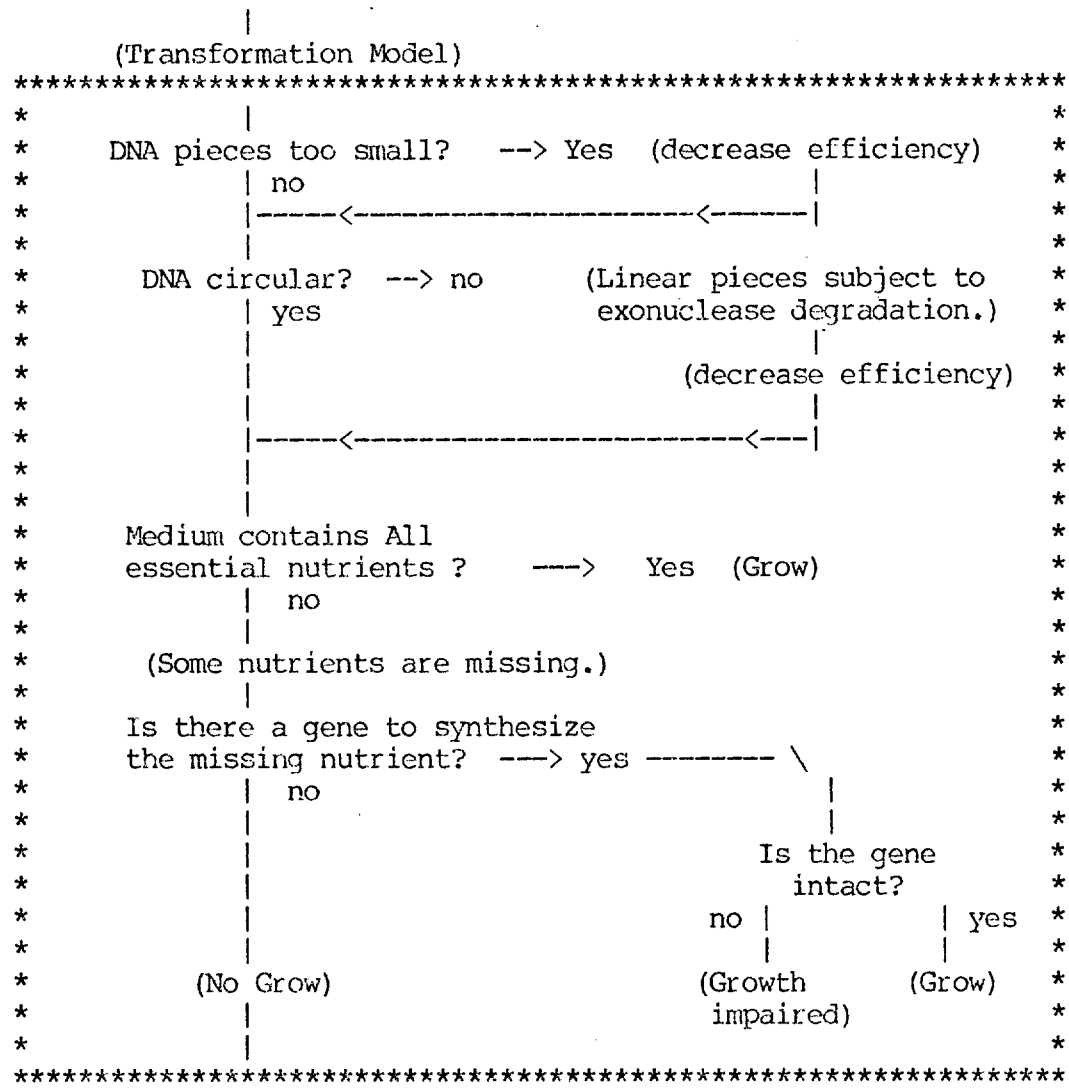
---

(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	ø	ø	ø
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.