Report 78-12
Stanford -- KSL

The Design and Evaluation of the First Crysalis System.

Allan Terry, Robert S. Engelmore,

Jul 1978

Stanford Heuristic Programming Project
Memo HPP-78-12

July 1978

THE DESIGN AND EVALUATION OF THE FIRST CRYSALIS SYSTEM

by

Allan Terry and Robert S. Engelmore (1)

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

The Design and Evaluation of the First CRYSALIS System

by Allan Terry and Robert S. Engelmore(1)

July 10, 1978

Abstract

This paper reports our experiences with the first version of the CRYSALIS system. The current state of the system and plans for revision are summarized. We have found that a design (in the software engineering sense) is a valuable tool for the evaluation and augmentation of a program, even when the design is done ex post facto. Using such a design, we discuss the major flaws of the existing system and how they could be corrected. Finally, we show how the architecture of this system could be useful for certain other task domains.

---

(1) Supported by the National Science Foundation (MCS74-23461 A01) and by the Advanced Research Projects Agency (MDA 903-77-C-0322)

Table of Contents

Section / Subsection		Page
1.	Introduction	1
2.	A Quick Introduction to Software Design	2
3.	State of the First System as of December 1977	5
4.	Some Problems Encountered in the First System	6
5.	Lessons from the First System	9
6.	Conclusions	11
Appendix I		
	A Rationalized Design of the PROTEIN Program	12
Appendix II		
	Existing Knowledge Sources	35
References		37

1 Introduction

There is an old programming adage that says one should expect to throw away the first version of any non-trivial system. This paper is a "final report" on the first version of the CRYSALIS project. As such, we will summarize the current state of the system and show where we plan to go with it. We have found that a design (in the software engineering sense) is a valuable tool for the evaluation and augmentation of a program, even when the design is done ex post facto. Using such a design, we discuss the major flaws of the existing system and how to correct them. Finally, we show how the architecture of this system could be useful for certain other task domains. The design details and the current knowledge sources of CRYSALIS are presented in the appendices; the main body of this report is a commentary on the system.

Before examining the program we will say a little about our task for readers not familiar with it. Briefly, the goal of CRYSALIS is to produce a model of the three dimensional structure of a protein molecule. This program is given experimental data in the form of an electron density map (or suitable abstractions of one) which is derived from x-ray diffraction studies of the crystallized protein. The protein's composition is also given in terms of its amino acid sequence and any cofactors present. The task is to recognize and match features in the data with the known composition thereby establishing locations for the atoms in the hypothesized solution. For human crystallographers, this is a rather ill-defined problem of visual recognition of features guided by constraints and expectations generated by the known composition. For further details see [5].

During the evaluation of our first implementation of the CRYSALIS system, while trying to find what worked and what didn't, we discovered that a design document can be a useful tool. What we mean by "design document" is an explicit and formal exposition that includes the specifications of the program, the ideas and theoretical tools to be used in solving the assigned task, and the decisions taken to implement these ideas. In our case this means a top-down exposition of the design with all decisions explained or rationalized. Many projects in artificial intelligence have designs but in the sense that most programmers do give some thought to the overall structure of the program before starting to code. These informal, undocumented designs usually contain large gaps between the program requirements and implementation details, thus leaving many important decisions to be made at the terminal. We learned from our own experience that the design must be detailed and must be written or else the fallibility of memory destroys the value of having a design in the first place. In many places our implementation differed from our original intentions mostly because we had forgotten what these intentions were. Program evaluation, the need to understand exactly what we had done, and the

need to avoid similar mistakes in the next round of system building were other considerations which prompted the use of design techniques. We have reached a point in our research where we need to evaluate what worked and should be saved, what new features must be added, and what notable failures should be avoided in the future. When a program is written from a design this evaluation of theory and code becomes relatively objective, as the design acts as a guide to what ideas the system embodies and how they were translated into runnable code.

2 A Quick Introduction to Software Design

The obvious question is "What exactly is a design?" A design is a representation of the program which can be as informal or as detailed as the situation requires. This representation helps the system builder by providing a framework for organizing the mass of detail involved in any substantial program. A useful design includes more than this representation (the code itself can be thought of as the most specific design), it also documents the reasoning used to expand the initial idea into the final code. To make such a document one must gather information, consider alternatives and judge their feasibility. So to be useful, a design must contain the major decisions and why they were chosen over all the alternatives that might suggest themselves. It is important that this information not be lost, for the implementers will need it if there is any question about the intent of the designer; it is also invaluable information for the designer should revision become necessary. These uses lead to our definition of design: a design is an abstraction of the program which serves to guide the process of implementation (as an architectural blueprint serves a carpenter), by explaining how the program works, and by explaining why the particular solution was chosen through a structured documentation of the reasoning.

There is no consensus on what form a design should take. Years ago the alternatives were flowcharts and a high-level description in English. Now the software engineers have filled out the spectrum somewhat with such devices as Program Description Languages [2], HIPO charts [13], and structure charts [15]. Often the choice of design representation depends on the level of detail one is working at; structure charts are good at showing the overall form of a system while descriptions in some sort of metacode are better at showing the design of a module in detail. Our interest in design lies primarily with its use as a documentation of ideas and the higher-level views of the system's architecture it can provide. The apparent complexity of our system is not so much due to its size as it is to the interactions of its knowledge sources. If we were most interested in the detailed control structure of the interpretive program we would choose some

graphical design representation. In our case the reasons for the form of the control are as interesting as the control structure itself, so we need a representation that is more explanatory than charts and less implementation-specific than the various metacodes. Given this orientation, we choose to use "justified decision statements" [6][7] as the main representation of our design. Each time we come to a decision, whenever we can add to the emerging design, we state that decision in a sentence or two and then write a short paragraph that rationalizes that decision. This explanation should not only say why the choice was made, but also display the major alternatives and why they weren't chosen. This representation seems easy to write and seems to serve the uses of design given in the definition above. The decision statements themselves provide the core of the design while the justifications form the documentation of the reasons behind the choices.

Just as there is no single way to represent a design, there is no single way to construct one ([1] is a good survey of methods). We favor a top-down approach, because it seems to fit the type of exploration common in A.I. research. While many systems are implemented in a "both ends toward the middle" manner, with top level control being built in parallel with lowest level utility functions, we feel the design process is most often some variant of a top-down method(2). Having been inspired to solve a particular problem, the first task is to write problem specifications, probably the most neglected part of programming. These specifications should define the task and spell out any restrictions or constraints known before designing starts. This document needs to be explicit as it forms evaluation criteria for the design and because anything not specified is assumed to be left to the discretion of the designers. Although there is no set form for specifications, we have found the following topics useful:

GOALS — A brief statement of the problem domain and what the program is to accomplish.

INPUTS & OUTPUTS — This specifies form and content. The form of both inputs and outputs should not be detailed (unless I/O formats are critical) but stated functionally, so the designers know what is expected. For example, "Sentences will be input in conceptual dependency form".

BUILT-IN KNOWLEDGE — An important part of most AI systems is the knowledge built-in for problem solving. This need not

---

(2) Note that top-down design is not the same thing as top-down implementation. While neither activity may actually proceed in a strictly top-down manner, it is important to present the design as such for clarity.

be given in detail (it may not even be known at this point) but the areas it must cover should be mentioned. For a physics problem solver this might be: "The program must have available all the formulas and facts expected of a freshman physics student."

CONSTRAINTS — The constraints provide the bounds within which the system must work. Include all constraints to be placed on program efficiency, resources allocated, methods and algorithms to be used, and general "system ideology". An example here would be the constraint placed on a tutoring system that "Output must be understandable to sixth graders"

VALIDATION & TESTING — This section should detail how the final system is to be validated; the method should be given and the data too (if that is possible).

While the exact form and content of the design is tied to the needs of the problem, there are a few elements that seem to be general. When using a top-down method these elements arise from the order in which the major decisions are made. The first level of the design should deal with the system as a whole. The overall architecture (linear, hierarchical, heterarchical, etc.) should be decided upon along with the major components needed to implement it. When this framework has been established, the designer can then fill it in, level by level. Each successive step in the design process involves the further detailing of the decisions in the previous step by decomposing them into smaller, more precisely specified entities. The major content of any design level is the body of decision statements and their justifications. In addition, it seems very helpful to include a list of the unanswered questions. This list can act as a focus for the next level and it gives a good indication of the limitations of the design. When the designer feels all pertinent items have been specified to the current level of detail and the current design seems coherent and correct, he or she then goes on to the next step. This process continues until coding can start or until the design is as detailed as required.

In reality this procedure is rarely carried out in such a strictly one-pass manner because the designer is not (usually) omniscient. Design is an iterative process of information gathering, where each level is as complete as possible with respect to what is known at that point. Often the elaboration of a decision acts as a test; what appears to be a good idea in the abstract becomes less attractive as more concrete details are considered. Feedback between levels of the design or between the design and the specifications are

important and should be allowed for because people make mistakes and learn from experience. In fact, this feedback is often counted on because many designers use probes to test their ideas or to see what might be needed. The designer often picks a small subsystem to work out in some detail, i.e. some part of the total system that uses the features to be tested. By designing this probe system several levels further than the current level one can get new information about the effects of a higher level design decision, or just see what features might be required.

The common programming practice is to design a program, then code and test it. In our case, the design, as presented in Appendix I, was compiled ex post facto. A year elapsed between the initial implementation from informal design ideas and the documented design presented here. However, working from the original notes and program listings, we believe we have captured the principal ideas and motivations. The original architecture was specified during the summer of 1976 by Robert Engelmore and Penny Nii, who also did some of the initial implementation. In Oct. 1976 Allan Terry took over the LISP part of the system, and Engelmore started working on improving the data representations and access functions.

3 State of the First System as of December 1977

Even though there was no written design to guide our implementation, we had an informal design that served fairly well. In comparing the system to the design given in Appendix I, we find that the system embodies the design in all major ways but two: the addition of certainty factors and an agenda mechanism. The informal design was based on the similarities between the requirements of this system and one which was developed for a military signal processing task [14]); Our original thought for this project was to fit this new task into a nearly identical architecture and see if it worked. The time-based features for signal processing were dropped, some system details were changed or added, but basically the structure of the signal processing system provided our first order design.

The knowledge sources currently in the system are given in Appendix II. The development and implementation of these knowledge sources has been slow, mainly because we had to become our own experts in some major ways. Most of our data representations are not widely used in the crystallographic community so we must infer many of the rules ourselves. Most of the implemented knowledge sources are concerned with toeholds, i.e. islands of certainty from which to expand the hypothesis. Finding toeholds in a molecule seems to flow logically from finding large atoms to finding cofactors that contain large atoms

to finding more remote toeholds such as the two ends of the polypeptide. In addition, the system predicts (poorly) regions of helical structure by using the method of Chou & Fasman [3], and tries to find what we call "interesting sequences". These are patterns of sidechains found in the given amino acid sequence that ought to be easy to see in the map, e.g. patterns of exceptionally large or small sidechains. The system is capable of working on globular proteins of moderate length and with data of resolution in the range 2.0 to 2.5 angstroms. It can find most of the toeholds humans use (except disulphide bridges) and will hazard a guess as to some of the secondary structure. In addition to the knowledge sources listed in Appendix II, we have developed some FORTRAN routines such as sidechain template matchers and a location generator for peptide planes that form an alpha helix. These routines have not been incorporated into any knowledge source yet because the system has not advanced enough to make use of them.

Two proteins were used as test data. The first was rubredoxin, a protein of 54 residues and a cofactor consisting of an iron atom tetrahedrally coordinated by four cystine sulphurs (see [5] for a glossary of crystallographic terms). For our test purposes, we generated a 2.0 angstrom resolution map from published atomic coordinates; we did not start from the experimentally derived electron density map. A skeleton [9] was calculated on a grid spacing of 1.0 angstrom. Rubredoxin is a small protein and its structure is one of the best known, so we consider our skeleton to be as clean and informative as we will ever see. The second protein is cytochrome C2, a medium sized (112 residues) protein with several alpha helices and a heme group as cofactor. The map is also a calculated map but only to 2.3 angstrom resolution which means it has only 2/3 the information content as a 2.0 angstrom map. Despite the poorer resolution (which also degrades the skeleton) the particular geometry between the heme region and the helix at the beginning of the protein chain makes this protein structurally interesting.

4 Some Problems Encountered in the First System

When new problems started accumulating faster than solutions to old ones we decided to evaluate the system. Few major projects can avoid a pause to review progress but fewer still have a neat algorithmic method for doing it. What generally happens is that bugs are found and patched for some time until the suspicion arises that the bugs have a deeper cause. At some point, debugging becomes cosmetic, allowing the program to continue running, but not addressing the real problems. When this occurs one must evaluate the original design decisions in the light of implementation experience. There are two

kinds of faults that can surface during this review, faults of implementation and more interestingly, faults of design. Unfortunately the two are often intertwined, as when the implementation comes out badly because the design isn't specific enough in its intentions. In our case there is another complication: not everything designed was implemented because we chose to work with a subset of features to see if that would be sufficient. Although we have nowhere near a full complement of knowledge sources, we feel the ones we do have are a fair test of the design in that they exercise the main features of the proposed system. In this section we display the more basic faults encountered in our first system, using the design as a guide to their evaluation and correction.

Syntax of Rulesets. The most obvious problem is that the rulesets leave much to be desired. Our rules are hard to read, they abound with LISP functions and domain-dependent details. In part this is bad implementation - we should have devoted more time to developing a good set of access functions to hide such details. In part this is due to inexperience, as we didn't realize our syntax was inadequate until we used it. In the final analysis our awkward rulesets should also be blamed on insufficient design. For one thing we left the explanation subsystem only vaguely specified (witness II.D and III.D.3), leaving hooks to hang it on later. The rule syntax given in III.A.5 specifies the general form of the rules but the design is not clear about what specific objects form the vocabulary of these rules. With this not given and with no clear idea what we wanted to explain, the rules drifted from a high-level description of the application of knowledge to a more LISP level of expression. A further contribution to the general confusion is the use of the bindings section. We specified in point III.A.2 of the design that bindings could create local variables but we didn't specify in detail how they were to be used. As a result we were seduced by the "convenience" of setting local variables in rules rather than writing cleaner rules. We should have restricted local variables to being set only in the binding section using system access functions or simple LISP. Instead of writing rules in terms of a series of intermediate values, we should use a better set of access functions in conjunction with these restricted bindings to allow the rules to express more meaningful chunks of knowledge. This problem is a good illustration of the utility of design. Without one we produced some rather fuzzy specifications of what we wanted. Part of the idea of a formal design phase is that its discipline and explicitness forces one to recognize incomplete ideas and deal with them before coding.

The Role of Strategy. The next major problem has several symptoms, the first being the paucity of strategy rules. The strategy ruleset encodes a jumble of purposes including an initialization rule, a termination rule, two rules to run the system machinery, and only one real strategy rule that examines the eventlist. It could be that the system is still too small to need strategy but we think it is more

likely a design fault. One piece of the problem is that the design states (III.C.3) that only the first event is matched by the eventrules. This is adequate for implementing depth-first or breadth-first searches but it now seems that the strategy rules as designed serve as a second eventrule set with more complex matching than the first. Another piece of the problem is that events are matched by name (also III.C.3) rather than by the actual change in the blackboard. As a result the sequence of computation is sometimes held together by a series of meaningless programming tags rather than something that relates to the task. The problem is that the meaning and role of events was not clearly thought out. It isn't clear if events are to distinguish between expectations from the sequence or results from the data. It can also be seen that the conception of strategy in II.C.1,3 of the design is different from the lower levels at III.C.6. The strategy ruleset was narrowed in function from an overall problem strategy selector to an auxiliary eventrule set. The fix here is to redesign so that the strategy deals with problem solving tactics and the overall shape of the solution while the eventrules actually decide which knowledge sources are needed for the current task.

The Role of Events. The two problems above are examples of insufficient design, i.e. fuzzy thinking leading to program problems. This confusion over the role of events brings up an example of how design can guide later additions to the system. In the original design (III.C.8) it is specified that when an event is not matched it is to be requeued. This resulted in endless loops of unmatchable events when there are no strategy rules to do something with these events and no knowledge sources to consume them. Also, there are times when one wants some knowledge source to execute but there is no event to trigger it (this can occur when one line of exploration runs out of steam and the system must try a new line). These two problems prompted us to institute an agenda mechanism. The agenda starts with all the events that serve only to fire exploratory rulesets, i.e. events not created by any ruleset and so events with no associated hypothesis. If we had used a design we would have foreseen trouble, because this is a use of events not called for or integrated into the design. The real problem is the one mentioned above, poor understanding of the role of events. As it is, there are no theoretical underpinnings for the agenda. It is a fix for the symptom instead of the problem. The solution is to fix things at the design level rather than in the code; re-evaluate III.C.8 or augment the design.

The Role of Hypotheses. The last item to examine is the use of the hypothesis. We designed one particular organization for the hypothesis but had a lot of questions left open. It became evident as the program was being developed that the use of the hypothesis was getting too ad hoc. Almost every time we coded a new knowledge source it seemed we had to add new properties to some hypothesis element. While it is fine to do this upon need (e.g. we had to add certainty factors so that there

would be a way of determining which of the current hypotheses is best), it is against the spirit of a blackboard to use special purpose properties that exist mostly so one knowledge source can talk directly to another (this is explicit in I.5.7). If we had written a design before we started, the number of items in the open questions sections would have alerted us to watch for this kind of problem.

5 Lessons from the First System

Despite the faults mentioned above, it is encouraging that the system achieves most of what we expected from a first try. The basic SU/P architecture (discussed in [14] or in the design in Appendix I) has proved to be a useful framework for working on the application. The system meets most of the constraints listed in the problem specifications, and uses the knowledge sources it has in a reasonable manner. This first system was to serve mainly to debug our ideas and to test their feasibility. To this end we designed (informally) and implemented a minimal system that still exercised our ideas. This system uses only rudimentary event matching and only the data-driven mode of processing. While it appears that the ideas are sound (barring some of the bugs discussed in the last section), our minimal control structure is too restrictive. Besides using this experience to refine our initial ideas, what we do need to add is a better mechanism for focusing the attention of the system for more efficient problem solving and a way for the sequence (our model of the solution) to direct the processing.

Although we first thought that the system could be primarily data-driven, our experience is that such an approach alone will not be sufficient. A purely data-driven approach finds the initial toeholds such as the cofactor and the particularly prominent peaks but then the data stop giving guidance. Our basic source of data is the electron density map, 200,000 to 500,000 intensity values, of which perhaps 5% to 10% are significant. Each intensity value represents an average taken over a volume determined by the resolution of the data and the chosen grid spacing for sampling. Since the electron density map is a discrete sampling of a continuous object, derived from relatively poor resolution diffraction data and inaccurate phases, we must cope with spatial errors. Peaks, for example, can be a few grid points away from where they "ought" to be. There is a correspondence between intensity and actual electron density but the quality of the data is usually such that peaks may not lie on atomic centers and similarly, low intensity can appear where there are atoms. Our goal is the first and only solution that fits the data. This goal, the lack of an adequate hypothesis generator, and what we knew of crystallographer's methods led us to a primarily data-driven system. We now see that this is not

a suitable solution. The sheer quantity of the data and its low predictive power suggests a more strongly expectation-driven approach, i.e. using the data to confirm hypotheses instead of to propose them. There is a complete model of the solution in the form of the given amino acid sequence but it is weak in that it cannot predict any atomic locations. Its usefulness arises when it is anchored to the data by located toeholds. The model can then provide very local expectations of what to look for and some idea of where to look. By providing this kind of data selection the sequence greatly constrains the search. We feel this expectation-driven approach will become the dominant problem solving method of the system. Initialization of the solution occurs when the system finds the first objects in a data-driven manner, then when the sequence is anchored in a few places the system can expand these toeholds by a process of expect and verify (one should note that this is not the same as generate and test because specific atomic locations may not be given in the expectation, so that only an approximate hypothesis is generated).

One somewhat unexpected result is that the SU/P structure promises to be quite useful in domains where a set of heuristics have not been developed a priori. We have found that it is easy to add and debug new knowledge sources because of the modularity provided by the control structure. The system will run with any set of knowledge sources and data and will give whatever partial results it has rather than mutely failing. The control structure itself does not limit the user to any particular type of processing or strategy. Although we do not use backward chaining currently, the inference structure could be either forward or backward chained, and the solution can be expectation-driven, data-driven, or something in between. For tasks where the solution can be hierarchically decomposed as is our blackboard and where there is no useable hypothesis generator this architecture could be a good means for acquiring and debugging a set of rules. Once this knowledge base has been gathered the program can then be used in a performance mode or the knowledge can be compiled into some more algorithmic form for efficiency.

One warning must be made about AI systems intended for use as tools for working scientists; to be successful they must be especially well engineered. Use of our program structure in system building mode requires a high rate of interaction so if the user doesn't understand something or finds any part of the system hard to use, the temptation will be to revert to familiar old methods instead of this difficult new one. In our case the system will never be completed, for even after we "finish" it and turn it over to crystallographers, new proteins and new techniques will mean new rules to add to the knowledge base. The program will be developing over a period of years and by many programmers, so some steps must be taken to ensure they all have the same understanding of the system. A rationalized design document is an important tool for communicating this information.

6 Conclusions

We have given a brief view of a design methodology applied to a developing system in the field of artificial intelligence. We have also presented a design for the system which is used to evaluate the success of the current implementation. The next step in our research will be to use these results to design and then code a revised and augmented version of the CRYSALIS program. There are two aims in this. The first is to complete a program for the interpretation of electron density maps of proteins. The second is to explore the application of software engineering techniques to artificial intelligence. It is our hope that the use of design methodologies will improve the quality of AI programs and their documentation. Programs derived from careful designs are more likely to serve as building blocks for even more powerful systems rather than demonstrations of ideas that will be thrown away after publication. We hope the use of design techniques will allow detailed evaluation of the "AI toolbox" and aid in the formulation of a theory to guide their application.

Appendix I

A Rationalized Design of the PROTEIN Program

System Specification

Given an electron density map and a sequence of amino acids, determine the three dimensional structure of the protein at least as far as a model suitable for input into a standard refinement program. The system is to discover the positions of as many major atoms in the protein as it can, and to indicate whatever secondary structure it finds.

Goals

A running, useful tool for crystallographers
To explore the system architecture needed to solve the problem:
this includes issues of strategy and planning
knowledge representation
knowledge source management
flexible focus of attention
explanation of results and actions

Constraints

Much of the knowledge needed must be discovered and codified
The data is too noisy to be relied on to guide the solution by itself
There is a model of the solution but it can provide guidance only in very local contexts
The system is to run with minimal interaction with user
Must take up a "reasonable" amount of time and memory on a computer of the order of the SUMEX PDP-10
The system's inputs and outputs must be understandable to a crystallographer rather than an A.I. expert
Must be easy to add/change knowledge
The system must be able to help the user understand and modify its knowledge by providing an explanation of its actions upon request
Need to have plausible "partial programs" and partial results along the way to justify continued research

Input Data

Electron density map is the most basic. This contains one-fourth to one-half million intensity points in three-space. The quality of this map can be described by two measures; its signal/noise ratio (low to medium) and its resolution (which is not good enough to resolve individual atoms).
Peak list - a list of local maxima in density map
Skeleton - fairly crude description of map in terms of nodes and links
Sequence - the known composition of the protein in terms of its sequence of amino acids and heterogens (if any)

Knowledge Available

CHEMICAL - properties of amino acids and cofactors, bonding patterns, allowed configurations, interactions with the solvent, possible secondary structures

EXPERIENTIAL - the expertise of the domain including:
how to weigh the data
heuristics for recognizing useful features in the data
strategies for focusing attention
methods for verifying hypotheses
the "common sense" of an expert in the domain

SYSTEM KNOWLEDGE - knowledge the system has about itself including
current state of the solution
what knowledge is currently available
data representations in use
history of the run so far

Outputs

A crude model of the protein structure, i.e. positions of as many atoms as possible to within about 0.75 angstrom along with any higher-level structure (such as helices) the system found
Some indication of completeness and trustworthiness of final model
A "lab notebook" of actions and results if the user so desires

Validation and Testing

The system can be validated by comparing its solutions against human

solutions of the same proteins. Fortunately, it should be fairly clear if the answer makes sense or not because crystallographers have many ways of evaluating their own solutions already. We will consider the system to be out of the experimental stage when it can locate 70% of the non-hydrogen atoms in the molecule to within 0.75 angstrom.

There are several intermediate goals the development of this system can aim for. In order of increasing difficulty they are:

Solution of small (~50 residues) protein of simple structure from 2.0A resolution map calculated from a known model

Solution of medium (~100 residues) protein of more complex structure from 2.0A Fc map

Solution of medium proteins with real data (i.e. an Fo map) at resolution of about 2.0A

Solution of moderately complex proteins from real data of resolutions up to 2.5A

I. System Design - First Level

Introduction

The purpose of the first level in the design is to make the most global decisions about the architecture of the emerging system. We will use the specifications just given to decide on the major components of the system and choose some of the more general features such as knowledge representation. This first level in the design will provide a framework of ideas to be elaborated and extended in later levels.

A. Major Components of System

The KNOWLEDGE BASE is the store of domain-specific facts and heuristics This provides the control system with knowledge to apply and the explanation system with data.

The BLACKBOARD is the central communication device, containing the current hypothesis.

The CONTROL selects knowledge and uses it to infer new results. It makes this selection on the basis of what it sees in the blackboard. New inferences are put on the blackboard for use in later inferences and on the history for use by the explanation subsystem.

The HISTORY is the database of the system's actions for use by the explanation subsystem.

The EXPLANATION subsystem uses the history to explain how and why the system used the knowledge it did.

B. First Level Design Decisions

1) Many of the design ideas of this system will be based on the designs of HEARSAY-II [11] and SU/X [14]
   Many of the basic characteristics of this task are similar to those in these systems
   Both systems were successful so capitalize on their experience

2) Data to be used are the peak list, skeleton, and segment list
   The map itself is too big to use - will avoid if possible, the hope is that the more abstract representations are sufficient
   The segment list is a compressed version of the skeleton
   Ridge-line graph structure to be used instead of skeleton if available

3) Will use 2A resolution maps, skeletons defined on grids of about 1A, and peaks defined on grids of about one-half angstrom
   Better resolution maps are too good to expect in practice, 2.0 - 2.5 A. is common range of good maps used by the crystallographers themselves. Any less resolution and the problem becomes too fuzzy for us to interpret because the inferences become too weak.
   A grid spacing of approximately one angstrom for the skeleton seems to be a good compromise between accuracy and sheer bulk of data.
   A grid spacing for the peaks of approximately one-half angstrom seems to be needed to define the heights and locations of local maxima with sufficient accuracy

4) The basic problem solving paradigm of the system will be hypothesize and test
   The knowledge is such that it can only provide small local inferences
   Given knowledge is inexact and heuristic so hypotheses derived from it must be checked for validity when created to limit the effects of error
   The domain is not known enough that there is a feasible legal or plausible "move generator". Because of this, no heuristic search method that relies on the global control provided by such a generator can be used

5) Main knowledge representation will be production rules
   We hope rules are easy to get from expert and easy for expert to understand
   Having knowledge in small, uniformly coded chunks should make it easier to manage
   Rules could provide the modularity needed for a constantly changing knowledge base
   Limited syntax of rules is amenable to automatic examination such as explanation requires

6) Control will be layered as strategy then rules
   Need strategy to make system more adaptable and efficient
   Strategy must control rule execution to prevent thrashing
   To deal with many proteins will need a flexible application of knowledge sources

7) Communication/coordination via a blackboard; a uniform representation of the emerging hypothesis and other global data
   Uniform communication between all parts of system reduces complexity and increases clarity
   The need to add and modify knowledge constantly implies that new units of knowledge should be independent from what already exists in the system. Anonymous uniform communication through a blackboard is one way to attain this independence
   HEARSAY-II had good experience with it

8) Keep only a current best hypothesis
   We feel solution process will be an incremental buildup of the hypothesis rather than a cycle of proposed global solutions, each rejected in turn until the correct one is found
   Don't need to keep large search tree around
   Don't have to accumulate lots of evidence and weigh it at each point

9) Hypothesis will be represented by objects at different levels of abstraction.
   Other systems (notably HEARSAY-II) have shown its utility
   If a natural decomposition can be found, these levels can provide a useful basis for alternate views of the hypothesis
   Makes explicit what level is being worked on and what knowledge is applicable

10) Explanation will be based on a history list of inferences made by the system.
   This is patterned on the mechanism used successfully by MYCIN
   Must save inferences because they are what must be explained

11) If given a choice, do it the way people do it.
   This is more of a philosophy, pattern the problem solving strategy the way crystallographers solve proteins.

12) Default plan for solution: find cofactors first, then associated large atoms such as sulphurs, finally attempt to trace the backbone and find sidechains.
   This is a very high-level implicit strategy of being opportunistic - it says to find the most obvious things first.

13) At this point in our research we will concentrate on building a performance system, all other adjuncts such as automatic rule acquisition will be left until this has been accomplished

C. Open questions

1) On data

We will lean heavily on the skeleton but there is nobody else using it except Greer. How good is it?
How true is the topology? How unambiguous is it?
What features should we use it for?
What information is there in the peak list?
How much do we lose in not using the density map itself?
What is characteristic of data of the resolutions chosen?
How can the system be designed to accept new representations at a later date and not be bound to the current ones?

2) Knowledge representation

What flavor production system?
Some things seem better as tabular information, should everything be in rule form?
Can a restricted enough syntax be found so that the system can automatically manipulate the rules?
Are the rules really independent?

3) Control cycle

What will the control cycle be?
What role will the strategy have, what powers?

4) Focus of attention problem

Any knowledge source could theoretically read the entire blackboard, but very few should. How to efficiently direct the system's attention to the subproblem that is most important at the moment?

5) Current best hypothesis

Do we lose any information or efficiency by going this route?
How often will hypothesis elements later turn out to be wrong?
If a proposed hypothesis element is wrong, what should be done with any other elements that depend on it?

6) Hypothesis vs data

Both the data and the hypothesis are hierarchies, is there any problem with the fact that they overlap a bit semantically?

7) Explanation

Is more explanation needed than what a simple history list can provide?
Is a history list sufficient in a system that is not limited to a backward chained regime?
Is there a better mechanism than that used in MYCIN?

8) How much can this system be patterned after humans?

It seems this system will use different representations and will not do its reasoning from a visual base as people do.

9) How to acquire and incorporate the knowledge of experts.

10) How much human intervention should there be?
There is a spectrum of choices for the degree on intervention. This system could attempt to do everything by itself (barring temporary use of people as "stubs" for missing knowledge) or at the other extreme, the system could be only a smart lab assistant that needs guidance fairly often. The answer to this depends on the amount and quality of knowledge we can discover and build in.

II. Second Level Design

Now that the major components have been defined and the major decisions have been made, the task is to specify the components in more detail. The first level design points out many theoretical questions but leaves their implementation open. This level should address these problems rather than implementation issues. Some of the design decisions at this level will overlap those of the first level; this is done so that they can be developed in greater detail and so that the design at this level is more of a logical whole. Decisions that are logical consequences of previous decisions will not, in general, be explained or justified.

A. The Knowledge Base

1) This section elaborates on what kind of knowledge the system will have and how it will be represented; questions of how it is to be used will be deferred until the section on control. Of the given constraints, the most important here is the need for the system to grow incrementally by the addition of new knowledge. Without the ability to modify its knowledge the system cannot be written since most of the knowledge is not known yet and can only be developed and tested incrementally. Also important but somewhat secondary is the constraint that the system be able to explain itself.

2) There will be three kinds of knowledge:
   1) Domain facts such as are found in textbooks
   2) Inference heuristics - heuristics for using data, facts and the sequence to infer new hypothesis elements
   3) Strategy heuristics - when to apply the first two kinds of knowledge to obtain a solution

3) The "physical reality" of the system need only be a subset of the world of chemistry at the atomic level.

The main concepts needed are those of bonds, atoms, and their interactions. The system will also need to know about properties of amino acids and their role in protein structure; this includes such things as interaction with the solvent and with each other, bonding patterns, and known secondary structures.

4) The heuristic knowledge of the domain will be represented as production rules.

A production rule representation of knowledge is chosen here because it seems to be the most natural and flexible representation that satisfies the given constraints. A rule-based system is easy to expand since the rules, if well written, are highly modular chunks of knowledge. Being small and of standard format, they are handy units for the explanation. Finally, the heuristics we have gathered so far are already very close to rule form. The reader is referred to [4] for further discussion of production rules and their advantages.

5) Gather production rules into sets of rules called knowledge sources (KS's).

The bulk of the knowledge in the system will be in inference rules. At this point it isn't clear how many rules there will be but probably on the order of hundreds rather than tens of rules. Although in the original versions of production systems all the rules formed a homogeneous set, the rules here seem to separate naturally into larger and independent chunks of knowledge. So, for efficiency and for ease of human understanding, gather rules into KS's.

6) Any fact that is used in only a very specific context can be represented as a rule - any facts that are used in a global manner can be represented statically in tabular form.

Some facts are used only once or have significance only within one rule. When that is true the clearest thing is to use the fact in that rule rather than go elsewhere to look it up. Some facts do not fit into any one ruleset. Such facts as basic properties of the amino acids are more efficiently stored as static facts accessible to the entire system. It would be possible to have an Amino.Acid.facts KS, but that seems somewhat kludgy in that such a KS would need a rule for everything that could be asked of it. Another consideration is that facts probably don't need much more explanation other than "by definition" or some such. If this is true then it doesn't hurt to look up that fact, and any such lookup function can trigger this canned explanation. The impact of this decision should be small because there probably will not be too many facts to represent outside of the rules.

B. The Blackboard

The general plan for this section is to first go into more detail about why a blackboard is needed, then move to what should be on it. The final set of decisions deals with how it will be used.

1) A blackboard mechanism will be used for communication and coordination.

In any system there will be communicating processes, the only question is how they will do so. Since one of the goals of this particular system is to allow for growth and change within the knowledge base, some way must be found to keep KS's anonymous. If communication is process to process as it is in most systems, the system must be aware of exactly what KS's it has by name. If information goes through the intermediate stage of a blackboard the goal of anonymity can be achieved. All information is then expressed in the common language of the blackboard and there is no need to have separate interfaces for each pair of KS's that want to communicate, nor is there any need to know the names of any KS.

2) The blackboard will contain the emerging hypothesis, the data, the amino acid sequence, and any globals required by the system.

3) There should be a uniform set of access functions to deal with the data and the chemical information (which includes the sequence).

These access functions are needed so that the rules can have a clean syntax with a minimum of data and task specific code. Other than this, there is nothing more to design on this issue. It remains for the implementation to decide what representation the data will have in core.

4) The hypothesis will include information at three levels; atomic (individual atoms), superatomic (small groups of atoms such as a heme group), and stereotypic (common secondary structures such as beta-sheets and alpha helices).

Since the goal of the system is a model of the protein structure, all that is really needed is a way of representing atoms and their locations. The other two levels are included because they provide useful groupings by forming the basis for interpretive theories at that level of abstraction. Our discussions with crystallographers indicate these three levels form a decomposition natural to the domain since all our rules can

be easily written in terms of these abstractions. The only other objects that the final model might need are interatomic bonds. Most bonds are implicit in the hypothesis, the rest (hydrogen and disulphide bonds, for example) can be parts of the atoms they span. Finally, the utility of a multi-level hypothesis structure has been demonstrated by HEARSAY II and by several vision systems.

5) The hypothesis will be a graph composed of nodes at the three levels given above along with links between nodes and links into the data.

The utility of a multi-level hypothesis is that it has structure, it enables the rules to deal with different levels of abstraction. A beta sheet is composed of residues which are composed of atoms. So, a graph structure of nodes and links allows use of this knowledge. Although the hypothesis is conceptually a hierarchy, a tree representation will not do because atoms could belong to many stereotypes. Links to data are also needed as they are the evidential support of the hypothesis.

6) Only a current best hypothesis will be carried along, i.e. hypothesis elements can be created and modified but never destroyed, nor can there be more than one hypothesis element referring to the same physical entity at the same level. All links are assumed to represent the best identifications possible given current knowledge.

There are several alternatives for the type of hypothesis. One approach is to have a generator of goal states within the set of all possible states of the system. This state-space approach was ruled out because it seemed that the states would be unduly complex and because not enough was known about the problem space to carve it into states in the first place. Another alternative is to accumulate evidence within a node, the idea being that if backup is needed the system can see how it got to the failure and can fix blame. It was decided that the utility of this idea does not exceed its overhead costs. The linear nature of a polypeptide and the foreknowledge of the sequence are powerful (although local) constraints once this sequence is anchored into the data at a few places. Since the known composition gives us all the atomic hypotheses and most of the superatomic, the only confusion is in the data links. Once an element is created, the sequence guarantees the object exists. The only competition possible is which data links are the best. We feel the sequence constrains in a local enough manner that changing one hypothesis element will induce few changes in other elements.

C. System Control

There are several constraints on the form of the control:
there will probably be many KS's and rules
KS's will vary widely in cost and accuracy
each new protein structure will present a new set of problems
in terms of features present, quality of data, etc.
use blackboard and knowledge structures as given above
must allow for incremental growth of the system

1) The type of control should not be fixed and invisible to the system, strategy should be able to pick the processing mode from the given set of allowable modes.

This is a statement that control should be visible and flexible within the production system format. The strategy needs to know what modes of processing are available so that it can choose the best mode for the current situation. The explanation subsystem needs to know these modes along with how they work and which is currently being used so that it knows how to use the history list.

2) Some focus mechanism will control the decision of which KS will execute next.

Individual rules are already gathered into KS's for efficiency and for ease of understanding. Since there is likely to be many such KS's something must pick which is to execute next; something must focus the system's attention to the proper KS and region of the blackboard to solve the current task. Thus, there needs to be a basic cycle of KS selection and activation.

3) The basic cycle of focus and KS execution will be monitored and controlled by the strategic knowledge.

Since the experts have problem solving strategies that make the search more efficient, the system should be able to use that knowledge. The benefit of this new layer of control is that it can provide some global context to the focus mechanism by means of large scale changes such as changing the mode of processing or by deciding some area of attention to be unprofitable. In this role, the global strategy can be relatively ignorant of individual KS's, its job is to monitor, the overall process of solution and to alter the order of subtasks according to its wider information. This control scheme can be thought of as a progressive narrowing of attention, a decrease of abstraction until some particular rule is chosen for execution.

4) Both strategic and focus knowledge will be expressed as rulesets.

This is in keeping with the overall philosophy of representing knowledge in rule form. The control flow then is the execution of rules at different levels of abstraction. First, the strategy rules are scanned and executed to provide global information to the focus rules. Then the cycle of focus rules selecting KS's runs until some logical end. Finally the cycle begins again with control reverting back to the strategy rules which can decide that the problem is solved or to cycle again. Our hope is that this hierarchical organization of rulesets will provide for orderly and timely use of differing levels of knowledge while giving the control the flexibility it needs to do the task.

5) The default processing mode of the system will be an opportunistic and data-driven mode.

One of the fundamental issues the control system must face is "Which KS to execute next?". There are many ways to solve this problem, witness the number of different kinds of search there are. Since this system is loosely patterned after the way human experts solve the problem it it will be assumed that an opportunistic mode will be followed unless there is good reason not to. This means that if some discovery can immediately trigger another KS, then that KS should execute. This local depth-first type of searching should not be the only mechanism because it can easily get into trouble, but human experience shows that it is worthwhile to explore at least the immediately reachable consequences of most inferences.

6) The basic quanta of information for the control system will be the event, a record of some discrete change in the hypothesis.

As HEARSAY has shown [11], a KS should not be allowed to execute unless its preconditions are met and its execution will accomplish something useful. In most cases the system will be in an opportunistic mode which means that if the preconditions of a KS are satisfied by execution of the last KS, it is useful. The question then becomes one of knowing when the preconditions are met. The only information the preconditions are interested in is what is in the hypothesis (it is possible globals such as map resolution will be tested too, but their values are usually fixed at the start of the run). So, if each KS can scan only some list of changes, it won't have to examine the entire blackboard. This also provides the strategy

rules a way of influencing execution; these rules could reorder and delete events.

7) Focus of attention will be handled by examination and management of events, i.e the working store for the strategy and focus rules will be a list of events.

The use of an eventlist promises to be a powerful device. If KS's are invoked solely on the basis of their preconditions matching events, then the eventlist is all the control needs. In some sense, the method of KS selection is the focus of attention and turning this mechanism into a ruleset that matches events to KS's allows for many different processing regimes. If the eventlist is a stack, the system performs opportunistic depth-first searching. If the eventlist acts as a queue the processing becomes breadth-first. Various best-first searching schemes could be easily implemented, either as strategy rules or as a function.

8) The system will run mostly on forward-chained inferences.

The deciding factor is that there is an enormous quantity of data that requires a fair amount of processing to extract useful information. It would be possible to set up goals for each amino acid in the sequence then backward chain but there is nothing to guide this search. If instead, obvious features in the data are noticed and the system can infer their identity in terms of position within the sequence, this imposes strong constraints on the rest of the solution.

D. Loose Ends

The explanation subsystem is called for in the specifications but is a luxury that has not elicited much interest at this stage in the project. We choose to leave its design and implementation to a later stage in the system development. There are a lot of interesting issues in the idea of a self-explaining program but we would rather get some reasonable system in operation before dealing with them.

E. Open Questions

1) How are facts to be used? They can be rules or static information but what form will these take and which facts will not be rules? How much should we rely on templates of common structures and on stereochemical knowledge considering the data is so noisy?

2) What will the KS's look like? What kind of processing needs to be accomplished and what objects will the rules manipulate?

3) A big question is what to do with failed hypotheses. It is possible that such a situation will not come up, but what happens when some hypothesis element is proven wrong (or at least some KS says it is)? The system will then have to decide what it believes and modify the current best hypothesis accordingly. Will backup be necessary in such situations?

4) Does the system need to represent bonds explicitly?

5) As the hypothesis is set up, each element represents one item. Is that sufficient or will we need disjunctive hypothesis elements or even more horrible beasts?

6) What objects will the hypothesis need and what will their attributes be?

7) Several modes of information processing were mentioned in the section on control. What processing modes will actually be needed?

8) At the moment we can't think of many strategy rules. Is it the case that there won't be many and strategy should be played down, or will there be lots and this conception of strategy is inadequate?

F. Summary

At this point, most of the broad theoretical issues have been decided upon and further design will be increasingly oriented towards implementation details. Before going on to the next level of the design, it would be helpful to review what the system looks like so far.

The bulk of the system will be composed of task-specific knowledge represented as production rules and these rules will be gathered into larger logical units called knowledge sources. Both strategic knowledge and hypothesis-building knowledge will also be so represented. In addition, there could be a small amount of data stored in globally accessible tabular form.

The blackboard will contain the hypothesis, the data, the sequence, and any globals needed. All inter-KS communication will go through this blackboard and will be expressed in the common form of hypothesis and events. The main constituent of the blackboard, the hypothesis, is conceptually structured as a hierarchy of three levels (atoms, superatoms, and stereotypes) but will actually be a graph structure in the system. There will be only one hypothesis element for each physical object and its links will represent the best hypothesis currently available. It should be noted that nodes on the blackboard represent objects in the model that must be linked into the data rather than HEARSAY's notion that areas of the blackboard are areas within the data that must be labeled with a hypothesis.

In an attempt to gain maximum flexibility with minimum complexity and overhead, we have chosen a hierarchical control regime. This is a tradeoff of some program complexity in return for faster and clearer access to rules in the knowledge base. At the top level is a ruleset that embodies all knowledge on the strategy of solving the problem. This ruleset can pass control to the eventrules which encode the information needed to select the most useful KS to execute next. Once this selected KS executes (possibly producing changes in the blackboard packaged as events) control reverts back up to the strategy rules. It is possible to obtain many modes of processing within this framework; modes such as forward/backward chained inference or depth/breadth/best first processing. We have chosen the predominant mode to be opportunistic forward chained inference because it seems to work for human crystallographers. This control cycle is fueled by a list of events which acts as a working store for the eventrules and strategy rules. Other modes of processing are obtained by altering the method of adding events to the eventlist and by changing the way they are matched off.

III. Third Level Design

A. The Knowledge Base

1) There will be two kinds of KS execution; single hit (execute only the first rule to match) and multiple hit (execute all rules that match on a single sweep through the KS).

The idea of a single pass through the KS suggests itself because that is how the rules we have are organized. The "standard" production system interpreter cycles through the rules until none match. This is suited to a ruleset that describes a continuing process whereas our KS's will probably be more like collections of inferences to be made if their preconditions are met. A multiple hit KS will be more a checklist of items than an active process description.

2) Knowledge sources can have local variables which would be set by a "bindings section" before execution of the KS.

This is mostly a matter of convenience and aesthetics. The rulesets we can envision seem to do a little preliminary calculation then make some inferences. The use of global variables for local intermediate results is bad programming practice. It would also be possible to write rules so that no intermediate results would be set, but only at the cost of having complex and unreadable rules.

3) The main effect of a KS will be to change the hypothesis. This will be expressed by a function "CHANGEHYPO" that takes an old hypothesis element (or creates a new one) and puts properties on it.

Since the system uses a current best hypothesis there is no need to modify a hypothesis element, only to create or replace them. CHANGEHYPO covers all this activity and can associate such a change with an event name.

4) The form of a KS will be as follows: overall there is an optional bindings section which is followed by the rules. Bindings look like PROG bindings except that each variable must be set. The rules have a LISP-predicate-expression as a condition and then one or more actions. An action can be a call to CHANGEHYPO, a ruleset name, or some LISP expression. Comments are definitely allowed.

5) The formal syntax for rulesets is as follows:

<Ruleset> ::= (NIL (<Rule>$^{+}$)) | ((<Binding>$^{+}$)(Rule)$^{+}$))
<Binding> ::= (<Atom> <LISP Expr.>)

```
<Rule>          ::= ( <Situation> <Action>+ ) | <Comment>
<Comment>       ::= (* <Text> )
<Situation>     ::= <LISP Expr>+
<Action>        ::= <Ruleset Name> | <LISP Expr.> | <New Event>
<Ruleset Name>  ::= <Atom bound to a ruleset>
<New Event>     ::= (MAKE.EVENT <Eventname> <Hypo> <Attval list>)
<Eventname>     ::= COFACTOR | HEAVYATOM | etc.
<Hypo>          ::= <Atom> | (GENATOM) | (GENSUPATOM) | (GENSTEREO)
<Attval list>   ::= NIL | (<Name> <Value>+)
```

Where GENATOM, etc. generate new hypothesis elements of stated type.

6) The tabular knowledge will be kept in a property list form and will consist of a file of the twenty amino acids and the common cofactors. The information stored will be standard physical properties used in model building such as standard configuration, bond torsions, length and chemical properties such as atomic weight or acidity.

7) If it would be easier and more appropriate, allow a KS to call a FORTRAN program.

> Some number crunching such as template matching will be necessary and LISP just isn't suitable. FORTRAN is specified because that is the language crystallographers use most. A stylistic note: if FORTRAN is to be used, hide it in an access function, don't put it directly in a rule or it will be both unexplainable and very confusing to read.

B. The Blackboard

1) A model of the solution is given by the given amino acid sequence along with the structure provided by the stored information on amino acids.

2) The globals used will be: the tabular knowledge, data parameters such as resolution or space group of the crystal, system information such as the eventlist or the history, and a few globals set within KS's.

3) Atoms, being the most important part of the hypothesis, will have the following attributes:

TYPE	Type of atom, i.e. C, N, Fe, etc.
NAME	Specific name such as SD-23
BELONGSTO	Which residue does it belong to
SKELNODES	Any skelnodes which might be associated with the atom
PEAKS	Peaks identified with the atom

SPACE.LOC	Hypothesized actual coordinates of atom
MEMBEROF	Contains links to any superatoms or stereotypes this atom is part of
BONDED.TO	Some residue name such as Cys-123

4) Superatoms are considered to be aggregates of atoms and can have:

TYPE	Can include sidechain, residue, cofactor ...
NAME	As above
MEMBERS	The atoms that form the superatom
MEMBEROF	Link to stereotypes this is part of
BELONGS.TO	Residue that contains this superatom

5) Stereotypes are groupings of superatoms

TYPE	Alpha-helix, beta-sheet, backbone ...
MEMBERS	Links to superatoms
SEQUENCE	Residues that it is composed of

6) All three kinds of hypothesis element will have special purpose attributes specific to the given TYPE and level in the blackboard.

7) In order to encode a current best hypothesis, assume that the first value of any attribute is the best. The rest of the values are weaker possibilities that haven't been erased in case they might become useful.

C. System Control

1) Although a lot of features have been designed into the control so far, we will adopt a policy of using only those features actually needed, and leaving the rest to be inserted when they become useful.

We have very few KS's at the moment and don't wish to build an enormous and general system until it is justified. We hope that enough hooks and spaces have been left in the system that this insertion should be easy.

2) An event will have an event-name, an associated hypothesis element, and all the new attributes and their values. In this way changes can be packaged and named for future use.

3) Eventrule matching will be by event name and will examine only the first item of the eventlist.

At this point there seems to be no reason to do anything more complicated. No event is any more important

than any other so might as well settle for the easiest method. It would be preferable to call KS's by what they can do rather than by name, but we don't know how to do that yet and this mechanism seems adequate for the job. The actual event matched by the eventrules will become the focus of the invoked KS. A system global CURRENTFOCUS will hold this event.

4) The workings of the eventrules can be specified in some detail now. The eventrules will be a single hit ruleset with no bindings. The rules themselves will be a collection of event-name KS-name pairs. When the strategy rules passes control to the eventrules, the list is scanned until either some event name matches or until the scan fails (in which case some error processing must be done). If the event matches a rule, the KS named is passed the event to work on and is then executed. Given that the eventrules are just to invoke KS's there is no need for bindings. There is no particular reason why the ruleset must be single hit, the decision was to return to the strategy after each KS.

5) The strategy rules will be a cyclic ruleset.

These rules are the top level of the system and, unlike the other rulesets, control a continuing process. Since the system must process everything it can until it stops of its own accord, this ruleset must be cyclic. In terms of actual workings, the rules will be scanned until one fires. When control returns to the strategy, the interpreter will start at the top again instead of at the rule where it left off. What this means is that the ordering of the rules encodes information.

6) The strategy rules will have three functions. The first is to reorder and merge events in the eventlist before letting the eventrules work on it. Second is to pass control to the eventlist or the joblist. The final task is to decide when to quit. Until we come up with a better criterion, this will be when the system runs out of things to do.

7) The strategy rules will be a cyclic ruleset with no bindings. The condition part of a rule can include predicates for examining the eventlist or can be some LISP predicate. There can be more than one action and these include invoking an interpreter, merging two events, reordering the eventlist, calling a KS, or some LISP function.

8) If some event isn't matched by the eventrules or if the event has no hypothesis element, then requeue it on the eventlist.

The idea is that the system doesn't know what to do with that event now, but it is possible it will be merged into a useful event later. Such events should be kept in

case they become useful but they should be prevented from popping up at the front of the list for a while. This delay can be achieved by queueing the offending event since processing has been set as FIFO.

9) The system will have a dummy KS to take the place of any KS's the user will have but hasn't coded yet. The action of this dummy will simply be to remove its invoking event from the eventlist. With this facility the system can be tested when only partially coded, the dummy allows for events that have no follow-up KS yet.

10) There should be an interpreter for each processing mode used by the system.

This allows the strategy rules to change mode merely by calling a different interpreter. There is a tendency to try to write all the different kinds of rulesets in such a way that one interpreter will do for all. This is compact but does not allow for independent development and addition of new modes, and doesn't make much allowance for the fact that different processing modes might work in very different manners.

D. Explanation System and Other Loose Ends

1) A list of linked events will be kept as the basic data for the explanation system. Events represent changes in the hypothesis so they are a convenient packaging of what must be explained. We feel that if the system can explain the events and their ordering, we have a sufficient explanation.

2) When an event is added to the history, it must be marked as the successor of the event that begot it. It must also record which KS and which specific rule created the event so that the rule can be printed out as an explanation.

3) This machinery can be implemented as part of CHANGEHYPO and the eventrules interpreter. This is as far as we will specify the explanation for now, the idea is to leave hooks for it to be put in later.

4) The system should be provided with a user interface to simplify use and debugging. This will include a trace facility, a method of causing a break on some rule (since LISP will only break functions), and some way of stopping a run in the middle and saving the context.

E. Open Questions

1) The exact nature of the explanation needs to be worked out. What will be explained and in what format. The system as it is is a unified entity, the user does not interact and select between subsystems. How then will the user get into explanation mode? Another problem is that the decisions above assume the history is a well behaved tree structure but that might not be true if the system has lots of processing modes and switches between them often.

2) Another problem with having many processing modes is that they might require different kinds of KS structure. What other kinds of KS will be needed? It may be possible to write a KS so that it can be run in forward or backward chained mode, but it seems unlikely that it will be easy.

3) The theoretical ideal is to invoke KS's anonymously in terms of what they can do instead of their names. This would require some way for the system to read its own KS's to see what their capabilities are, a difficult problem. Note that this self-examination is really necessary, using a programmer-defined summary of a KS is no better than using the name.

Appendix II

Existing Knowledge Sources

Knowledge Sources of the PROTEIN System

INITIALIZATION	Examines the given data. Looks through the model and posits any cofactors and large atoms it finds. This KS generally serves to prime the system by getting some initial hypothesis elements on the blackboard and generating a list of events
PROCESS.KNOWN.PLACES	Takes given data which can be coordinates of atoms or labels of peaks, does consistency checking and puts the information on the blackboard
LOCATE.COFACTOR	If the protein has a cofactor, attempts to find it in the peaklist using information on cofactor geometry in AACID. In the case of hemes this is done by a template search in the map. The task for non-heme iron cofactors is to find the peak most likely to be the iron atom.
REMOVE.COFACTOR	Further specify the cofactor by finding nodes in the skeleton that lie within the template area. When this is done, remove the cofactor from the seglist to make chain tracing easier.
HYPO.COORDINATED.PEAKS	Given a cofactor, try to find peaks for any coordinated atoms such as sulphur. This KS relies heavily on the geometry of the cofactor region. If some but not all of such peaks are found, predict the locations of the missing peaks from the model.
FIND.CHAINENDS	A single-strand protein with the cofactor removed should have two ends. This KS

attempts to find these ends within the seglist.

PREDICT.HELIX — This KS in an embodiment of the Chou and Fasman method of predicting helical regions.

FIND.INTERESTING.SEQ — Examines the model for toeholds in the form of sequences of sidechains that might be relatively easy to identify in the map. An example is a series of three ringed sidechains in a row

In addition, there is a group of KS's that exist as a preprocessor. These are fairly expensive and need be run only once per protein rather than every time a user attempts to solve the same protein.

LABEL.SEGMENTS — This KS does quasi-visual processing of the segment list. Based mostly on topology, it labels segments as sidechain, mainchain smallbridge, and bridge.

TRACESEGS — Embodies knowledge about tracing "clean" parts of the skeleton.

MESS.TRACE — Counterpart of TRACESEGS, knows how to trace in the overconnected regions of the skeleton and how to find bridges.

References

1. B. W. Boehm, "Software Design and Structuring", pp102-128 in E. Horowitz (ed.) **Practical Strategies for Developing Large Software Systems**, Addison-Wesley, 1975

2. S. H. Cain, and E. K. Gordon, "PDL - A Tool for Software Design", Proc. AFIPS NCC 1975, pp271-276

3. P. Y. Chou, and G. D. Fasman, "Prediction of Protein Conformation", Biochemistry Vol. 13, Num. 2 (Jan 15 1974), pp211-245

4. R. Davis, and J. King, "An Overview of Production Systems", SAIL Memo 271, Stanford Univ., Oct. 1975

5. R. S. Engelmore, and H. P. Nii, "A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data", Memo HPP-77-2, Stanford Univ., Feb. 1977

6. P. Freeman, "Toward Improved Review of Software Designs", Proc. Nat. Computer Conf. 1975, pp329-334

7. P. Freeman, "Software Design Representation: Analysis and Improvements", ICS Tech Report 81, Univ. of Calif., Irvine, May 1976

8. F. Hayes-Roth, and V. R. Lesser, "Focus of Attention in a Distributed Logic Speech Understanding System", CS Dept. Tech Report, Carnegie-Mellon Univ., Jan. 12 1976

9. J. Greer, "Three-Dimensional Pattern Recognition: An Approach to Automated Interpretation of Electron Density Maps of Proteins , J. Mol. Biol. (1974) 82, pp279-301