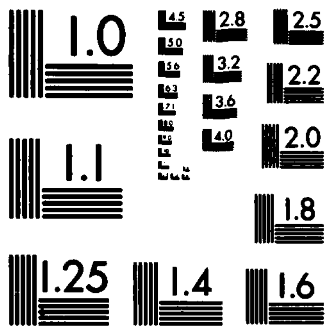
AD-A130 597 XPLAIN: A SYSTEM FOR CREATING AND EXPLAINING EXPERT CONSULTING PROGRAMS..(U) UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION S.. W R SWARTOUT 1/1

UNCLASSIFIED JUL 83 ISI/RS-83-4 PHS-1-P01-LM-03374-01 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A130597

ISI Reprint Series
ISI/RS-83-4
July 198_

(12)

University of Southern California

William R. Swartout

XPLAIN: A System for Creating and Explaining Expert Consulting Programs

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90291-6695

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ISI/RS-83-4	2. GOVT ACCESSION NO.: AD-A130577
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): XPLAIN: A System for Creating and Explaining Expert Consulting Programs	5. TYPE OF REPORT & PERIOD COVERED: Research Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): William R. Swartout	8. CONTRACT OR GRANT NUMBER(s): 1-P01-LM-03374-01
9. PERFORMING ORGANIZATION NAME AND ADDRESS: USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90291	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: National Institutes of Health, 9000 Rockville Pike, Bethesda, MD 20205	12. REPORT DATE: July 1983
	13. NUMBER OF PAGES: 58
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office): ----------	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

----

18. SUPPLEMENTARY NOTES

This report is a reprint of an article that will appear in *Artificial Intelligence*, a journal published by North-Holland, in late 1983 or early 1984.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

automatic programming, expert systems, explanation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (continued)

Traditional methods for explaining programs provide explanations by converting the code of the program or traces of its execution to English. While such methods can sometimes adequately explain program behavior, they typically cannot provide justifications for that behavior. That is, such systems cannot tell why what the system is doing is a reasonable thing to be doing. The problem is that the knowledge required to provide these justifications was used to produce the program but is itself not recorded as part of the code, and hence is unavailable.

The XPLAIN system uses an automatic programmer to generate a consulting program by refinement from abstract goals. The automatic programmer uses a domain model, consisting of descriptive facts about the application domain, and a set of domain principles which prescribe behavior and drive the refinement process forward. By examining the refinement structure created by the automatic programmer, XPLAIN provides justifications of the code. XPLAIN has been used to re-implement major portions of a Digitalis Therapy Advisor and provides superior explanations of its behavior.

ISI Reprint Series
ISI/RS-83-4
July 1983

University of Southern California

William R. Swartout

XPLAIN: A System for Creating and Explaining Expert Consulting Programs

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90291-6695

This research was performed while the author was at the Laboratory for Computer Science of the Massachusetts Institute of Technology and was supported (in part) by the National Institutes of Health Grant No. 1 P01 LM 03374-01 from the National Library of Medicine.

ISI Reprint Series

This report is one in a series of reprints of articles written by ISI research staff and published in professional journals and conference proceedings. For a complete list of ISI reports, write to

Document Distribution
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90291
USA

Contents

1. Introduction 1
  1.1 Major Points 2
  1.2 Outline 4

2. Digitalis Therapy and the Digitalis Advisor 4
  2.1 The Digitalis Therapy Advisor Testbed 5

3. Kinds of Questions 6

4. Previous Approaches to Explanation 7

5. Providing Justifications 10
  5.1 The Refinement Structure 12
  5.2 The Domain Model 13
  5.3 Domain Principles 14
  5.4 The Domain Rationale: More Detail 16
  5.5 Intertwining Specification and Implementation 18
  5.6 Integrating Program Fragments 19
  5.7 Assessing Toxicity 22

6. Generating Explanations 25
  6.1 XLMS Notation and the Phrase Generator 25
  6.2 The Answer Generators: Determining What to Say 28

7. A Discussion of the Automatic Programming Approach to Explanation 38
  7.1 Does Automatic Programming Affect the Performance Program? 38
  7.2 Is This Approach to Explanation Compatible with Others? 40
  7.3 Is Automatic Programming Too Hard? 40
  7.4 Is a Top-down Approach Really Necessary? 41
  7.5 Limitations and Extensions of the XPLAIN System 41

8. A Summary of Major Points 45

9. Acknowledgments 45

References 45

XPLAIN: A System for Creating and Explaining Expert Consulting Systems

Abstract

Traditional methods for explaining programs provide explanations by converting the code of the program or traces of its execution to English. While such methods can sometimes adequately explain program behavior, they typically cannot provide justifications for that behavior. That is, such systems cannot tell why what the system is doing is a reasonable thing to be doing. The problem is that the knowledge required to provide these justifications was used to produce the program but is itself not recorded as part of the code, and hence is unavailable.

The XPLAIN system uses an automatic programmer to generate a consulting program by refinement from abstract goals. The automatic programmer uses a domain model, consisting of descriptive facts about the application domain, and a set of domain principles which prescribe behavior and drive the refinement process forward. By examining the refinement structure created by the automatic programmer, XPLAIN provides justifications of the code. XPLAIN has been used to re-implement major portions of a Digitalis Therapy Advisor and provides superior explanations of its behavior.

1. Introduction

Computers can be inscrutable. To the layman, the computer is often regarded as either an omniscient, fathomless device or a convenient scapegoat. In part, this situation has arisen because computer systems are designed with little provision for self-description. That is, programs cannot explain or justify what they do. Typically, the designer of a data processing system can rely on a staff of computer support personnel to deal with problems and questions as they occur. However, even in relatively simple areas such as accounting and billing this approach has not been an overwhelming success, and it becomes less appropriate as we become more ambitious and attempt to use the computer to solve more sophisticated problems in an environment where support personnel may not be available to answer inquiries.

---

1. This report is a reprint of an article that will appear in *Artificial Intelligence*, a journal published by North-Holland, in late 1983 or early 1984.

Trust in a system is developed not only by the quality of its results, but also by clear description of how they were derived. This can be especially true when first working with an expert system. Expert systems are usually based on heuristics. While heuristics may provide good performance for most cases, there may be unusual cases where they produce erroneous results, or where the rationale for using them is faulty. If a user is suspicious of the advice he receives, he should be able to ask for a description of the methods employed and the reasons for employing them. In addition, the scope of expert systems, like that of human experts, is often quite narrow. An explanation facility can help a user discover when a system is being pushed beyond the bounds of its expertise.

As a further illustration of the need for explanation consider expert medical consultant programs.[2] In designing a consultant program, we must consider what sorts of capabilities we are trying to provide for the physician user. If we consider the interaction between a physician and a human consultant, we realize that it is not just a simple one-way exchange where the physician provides data and the consultant provides an answer in the form of a prescription or diagnosis. Rather, there is typically a lively dialog between the two. The physician may question whether some factor was considered or what effect a particular finding had on the final outcome and the expert is expected to be able to justify his answer and show that sound medical principles and knowledge were used to obtain it. Viewed in this light, we realize that a computer program which only collects data and provides a final answer will not be found acceptable by most physicians. In addition to providing diagnoses or prescriptions, a consultant program must be able to explain what it is doing and justify why it is doing it.

Researchers have recognized this, and many proposals for new expert systems have at least mentioned the need for explanation. Some systems have actually provided an explanatory facility. Yet existing approaches to explanation fail in some important ways.

1.1 Major Points

In particular, we will argue in this paper that current explanation methodologies are limited to describing what they did (or the reasoning path they followed to reach a conclusion), but they are incapable of justifying their actions (or reasoning paths). By justifications, we mean explanations that tell why an expert system's actions are reasonable in terms of principles of the domain—the reasoning *behind* the system. The knowledge

---

2. Some medical consultant programs include: MYCIN—a program that aids physicians with antimicrobial therapy [Shortliffe76], INTERNIST—a program that makes diagnoses in internal medicine [Pople77] and PIP—a program that makes diagnoses primarily in the area of renal disease [Pauker76].

required to provide justifications is not represented in typical expert systems because the program can perform correctly without it. Just as one can follow a recipe and bake a cake without ever knowing why the flour or baking powder is there, so too an expert system can deliver impressive performance without any representation of the reasoning underlying its rules or methods.

The reasoning that system designers go through during the creation of an expert system is precisely what is required to produce justifications, yet it is not likely to appear in the expert system itself. This paper presents one approach for capturing that reasoning. The basic idea is to use an automatic programmer to create the expert system. As it refines the performance program from abstract goals, the programmer leaves behind a trace of its reasoning, which can then be used by explanatory routines to justify the actions of the expert system.

In creating the expert system, the automatic programmer draws on two bodies of knowledge. The first, called the *domain model* contains the descriptive facts of the domain, such as causal relationships and classification hierarchies—the textbook knowledge of the domain. The second body of knowledge, the *domain principles* contains the methods and heuristics of the domain—the "how to" knowledge. The automatic programmer integrates these prescriptive principles together with the descriptive facts of the domain to produce the performance program. This process of integration is recorded and used as the justification of the expert system's behavior.

This explicit separation of prescriptive and descriptive knowledge is not made in most expert systems and has some important benefits in addition to facilitating justification. In systems where descriptive and prescriptive knowledge is implicitly intermixed, methods or rules must often be stated at an overly specific level of abstraction. This reduces the quality of the explanations the system can deliver. As we will see (in section 6) separating these different kinds of knowledge allows methods to be stated at a higher level of abstraction resulting in superior explanations. The other primary benefit is to modifiability: the domain model and domain principles can be modified (or used) independently. For example, most of the domain principles which were written to anticipate digitalis sensitivities can in fact be used to anticipate sensitivity to any drug. Only the domain model would have to change to allow these principles to be used in a new application domain. On the other hand, much of the domain model that was developed for correcting for digitalis sensitivities was also used in detecting digitalis toxicity. In expert systems that intermix descriptive and prescriptive knowledge, modification of methods can be quite difficult. The separation of the two eases that process.

1.2 Outline

The next section will describe the Digitalis Therapy Advisor, the program we have chosen as a testbed for our ideas about explanation, and some of the medical aspects of digitalis therapy. Section 3 discusses some of the kinds of questions that require explanation. Section 4 outlines some of the problems with previous approaches to explanation. Section 5 details how the automatic programmer works and section 6 shows how explanations are produced. Section 7 further discusses the problems and promise of this approach.

While we have concentrated on the problem of providing explanations to medical personnel, we do not feel that the need for explanation is limited to medicine nor that the techniques we have developed for explanation and justification are limited to medical applications. Medical programs provide a good testbed for the general problem of explaining a consulting program to the audience it is intended to serve.

2. Digitalis Therapy and the Digitalis Advisor

The digitalis glycosides are a group of drugs that were originally derived from the foxglove, a common flowering plant. Their principal effect is to strengthen and stabilize the heartbeat. In current practice, digitalis is prescribed chiefly to patients who show signs of *congestive heart failure* (CHF) or *conduction disturbances* of the heart. Congestive heart failure refers to the inability of the heart to provide the body with an adequate blood flow. This condition causes fluid to accumulate in the lungs and outer extremities and it is this aspect that gives rise to the term "congestive". Digitalis is useful in treating this condition because it increases the contractility of the heart, making it a more effective pump. A conduction disturbance appears as an arrhythmia, which is an unsteady or abnormally paced heartbeat. Digitalis tends to slow the conduction of electrical impulses through the conduction system of the heart, and thus steady certain types of arrhythmias. Due to the positive effect that digitalis has on the heart, it is one of the most commonly prescribed drugs in the United States.

Like many other drugs, digitalis can also be a poison if too much is administered. For a variety of reasons, including a small difference between a therapeutic and toxic dose, subtle signs of toxicity, and high interpatient variability, digitalis is difficult to administer. The physician must deal with the possibility that his patient may be more sensitive to the drug (for whatever reason) than the average patient. If a physician knows those factors that make a patient more sensitive he can reduce the likelihood of overdosing (or underdosing) the patient by adjusting the dose depending on whether he observes the sensitizing factors or not.

One possible toxic effect of digitalis is to increase the *automaticity* of the heart. In the normal heart, there is a place in the left atrium called the sino-atrial (SA) node, which sets the pace for the heart. Under certain circumstances, other parts of the heart can take over the pace-setting function. Sometimes this can be life-saving if, for example, the SA node is damaged. But at other times it can be life-threatening, since several pace-makers operating simultaneously tend to increase the likelihood of setting up a dangerous arrhythmia, such as ventricular fibrillation. When we say that digitalis increases the automaticity of the heart, we mean that digitalis increases the tendency of other parts of the heart to take over the pace-setting function from the SA node.

Over the years, a number of factors have been identified that also increase the automaticity of the heart. These include: a low level of serum potassium (hypokalemia), a high level of serum calcium (hypercalcemia), damage to the heart muscle (cardiomyopathy), and a recent myocardial infarction. When these exist in conjunction with digitalis administration, the automaticity can be increased substantially. We will concentrate on the first two factors in this paper.

2.1 The Digitalis Therapy Advisor Testbed

A few years ago, a Digitalis Therapy Advisor was developed at MIT by Pauker, Silverman, and Gorry [Silverman75, Gorry78]. This program was later revised and given a preliminary explanatory capability [Swartout77b]. The limitations of these explanations (and of those produced by similar techniques) will be discussed below. This program differed from earlier digitalis advisors [Peck73, Jelliffe70, Jelliffe72, Sheiner72] in two important respects. First, when formulating dosage schedules, it anticipated possible toxicity by taking into account the factors that increased digitalis sensitivity and reduced the dose when those factors were present. Second, the program made assessments of the toxic and therapeutic effects which actually occurred in the patient after receiving digitalis to formulate subsequent dosage recommendations. This program worked in an interactive fashion. The program would ask the physician for data about the patient and produce recommendations after those data were entered. When the dose of digitalis was being adjusted, the physician was asked to consult with the program again to assess the patient's response. This program was used as a testbed for this work in explanation and justification. In the remainder of the paper, this program will be referred to as the "old Digitalis Advisor".

3. Kinds of Questions

In the spring of 1979, a series of informal trials was conducted in an attempt to discover what kinds of questions occurred to medical personnel as they ran the Digitalis Advisor. In this trial, medical students and fellows were asked to run the program and ask questions (verbally) as they occurred to them. I attempted to answer these questions. The interactions were tape recorded and later transcribed.

No formal analysis of the data was attempted, but examination of the transcripts did suggest three major types of questions that might arise while running a consulting program. These were:

1. Questions about the methods the program employed:

Subject: "How do you calculate your body store goal? That's a little lower than I anticipated."

2. Justifications of the program's actions:

Subject: (perusing recommendations) "Why do we want to make a temporary reduction?

Experimenter: "We're anticipating surgery coming up and surgery, even non-cardiac surgery, can cause increased sensitivity to digitalis, so it wants to temporarily reduce the level of digitalis."

3. Questions involving confusion about the meaning of questions the system asked:

```
IS THE RENAL FUNCTION STABLE?
THE POSSIBILITIES ARE:
        1. STABLE
        2. UNSTABLE
ENTER SINGLE VALUE ====>
```

Subject: "Now this question...I'm not really sure...'renal function stable' does it mean stable abnormally or...because I mean, the patient's renal function is not normal but it's stable at the present time."

Experimenter: "That's what it means"

The first type of question can be answered by any system that can produce an English description of the code it executes. This sort of question could be answered by the explanation routines of the old Digitalis Advisor and by the XPLAIN system presented here. Answering the second type of question requires not only an ability to translate code to English but also requires that the system represent and be able to express the medical knowledge upon which that code is based. This medical knowledge may not be necessary for the successful execution of code, but is necessary to be able to justify it. The third type of question requires a system that can model potential differences between a user's understanding of a term and the system's and generate an explanation which reconciles the two. The XPLAIN system does not address this type of question.

4. Previous Approaches to Explanation

A number of different approaches have been taken to attempt to provide programs with an explanatory capability. The major approaches are 1) using previously prepared text to provide explanations and 2) producing explanations directly from the computer code and traces of its execution.

The simplest way to get a computer to answer questions about what it is doing is to anticipate the questions and store the answers as English text. Only the text that has been stored can be displayed. This is called *canned text*, and explanations produced by displaying canned text are called canned explanations. The simplest sorts of canned explanations are error messages. For example, a medical program designed to treat adults might print the following message if someone tried to use it to treat an infant:

```
THE PATIENT IS TOO YOUNG TO BE TREATED BY THIS PROGRAM.
```

It is relatively easy to get a small program to provide English explanations of its activity using

---

Fig. 1. How the Old Digitalis Advisor Checks Hypercalcemia

```
TO CHECK SENSITIVITY DUE TO CALCIUM I DO THE FOLLOWING STEPS:
1. I DO ONE OF THE FOLLOWING:
   1.1 IF EITHER THE LEVEL OF SERUM CALCIUM IS GREATER THAN 10 OR
   INTRAVENOUS CALCIUM IS GIVEN THEN I DO THE FOLLOWING SUBSTEPS:
      1.1.1 I SET THE FACTOR OF REDUCTION DUE TO HYPERCALCEMIA TO 0.75.
      1.1.2 I ADD HYPERCALCEMIA TO THE REASONS OF REDUCTION.
   1.2 OTHERWISE, I REMOVE HYPERCALCEMIA FROM THE REASONS OF REDUCTION AND
   SET THE FACTOR OF REDUCTION DUE TO HYPERCALCEMIA TO 1.00.
```

this canned text approach. After the program is written. canned text is associated with each part of the program explaining what that part of the program is doing. When the user wants to know what is going on, the computer merely displays the text associated with what it is doing at the moment.

There are several problems with the canned text approach. The fact that the program code and the text strings that explain that code can be changed independently makes it difficult to maintain consistency between what the program does and what it claims to do. Another problem is that all questions must be anticipated in advance and the programmer must provide answers for all the questions that the user might ask. For large systems, this is a nearly impossible task. Finally, the system has no conceptual model of what it is saying. That is, to the computer, one text string looks much like any other, regardless of the content of that string. Thus, it is difficult to use this approach to provide more advanced sorts of explanations such as suggesting analogies or giving explanations at different levels of abstraction.

Another approach is to produce explanations directly from the program. That is, the explanation routines examine the program that is executed. Then by performing relatively simple transformations on the code these explanation routines can produce explanations of how the system does things. This approach has been used to produce explanations in a number of systems [Davis76, Shortliffe76, Swartout77a,Swartout77b, Winograd71]. More recent work by Kastner, Weiss and Kulikowski [Kastner82] uses a mixture of this approach with the canned text approach to achieve good explanations in the limited domain of a precedence-based therapy selection algorithm.

The old Digitalis Advisor could examine the code it used to check for increased digitalis sensitivity caused by increased serum calcium and produce an explanation of what that code did (as shown in Figure 1). Like most similar systems, it also maintained an execution trace. The trace could be examined by the explanation routines to tell what was done for a particular patient. Figure 2 describes how the system checked for myxedema. The system also had a limited ability to explain why it was asking the user a question. Figure 3 shows the system's response when the user wants to know why he is being asked about serum calcium.

Since the explanation routines only perform simple transformations on the program code, the quality of the explanations produced in this manner depends to a great degree on how the system code is written. In particular, the basic structure of the program is not altered significantly, and the names of variables in the explanation are basically the same as those in the program. If the explanations are to be understandable, the expert system must be written so that its structure is easily understood by anyone familiar with its domain of expertise, and the variable and procedure names used in the program must represent concepts which are meaningful to the user.

Fig. 2. Explaining How Thyroid Function Was Checked

I CHECKED SENSITIVITY DUE TO THYROID-FUNCTION BY EXECUTING THE FOLLOWING STEPS:

1. I ASKED THE USER THE STATUS OF MYXEDEMA. THE USER RESPONDED THAT THE STATUS OF MYXEDEMA WAS PRESENT.

2. SINCE THE STATUS OF MYXEDEMA WAS PRESENT I DID THE FOLLOWING:

   2.1 I ADDED MYXEDEMA TO THE PRESENT AND CORRECTABLE CONDITIONS. THE PRESENT AND CORRECTABLE CONDITIONS THEN BECAME MYXEDEMA.

   2.2 I REMOVED MYXEDEMA FROM THE DEGRADABLE CONDITIONS. THE DEGRADABLE CONDITIONS THEN BECAME HYPOKALEMIA, HYPOXEMIA, CARDIOMYOPATHIES-MI, AND POTENTIAL POTASSIUM LOSS DUE TO DIURETICS.

   2.3 I SET THE FACTOR OF REDUCTION DUE TO MYXEDEMA TO 0.67. THE FACTOR OF REDUCTION DUE TO MYXEDEMA WAS PREVIOUSLY UNDETERMINED.

   2.4 I ADDED MYXEDEMA TO THE REASONS OF REDUCTION. THE REASONS OF REDUCTION THEN BECAME MYXEDEMA.

Fig. 3. Telling Why a Question is Asked

WHAT IS THE LEVEL OF SERUM CALCIUM?

ENTER NUMERICAL VALUE ====> why? *(The user types "why?" indicating that he questions why he is being asked to supply the value of serum potassium.)*

MY TOP GOAL IS TO BEGIN THERAPY. ONE STEP IN DOING THAT IS TO CHECK SENSITIVITIES. I AM NOW TRYING TO CHECK SENSITIVITY DUE TO CALCIUM.

This method of producing explanations has some advantages. It is relatively simple. If the right way of structuring the problem can be found, it does not impose too great a burden on the programmer; since the explanations reflect the code directly, consistency between explanation and code is assured.

Despite these advantages, there are some serious problems with this technique. It may be difficult or impossible to structure the program so that the user can easily understand it. The fact that every operation performed by the computer must be explicitly spelled out sometimes forces the programmer to program operations which a physician would perform without thinking about them. That problem is illustrated in Figure 2. Steps 2.1, 2.2, and 2.4 are somewhat mystifying. In fact, these steps are needed by the system so that it can record what sensitivities the patient had that made him more likely to develop digitalis toxicity. These steps are involved more with record keeping than with medical reasoning, but they must appear in the code so that the computer will remember why it made a reduction. Since they appear in the code, they are described by the explanation routines.

although they are more likely to confuse a physician-user than enlighten him. An additional problem is that it is difficult to get an overview of what is really going on here. While the system is explicit about record keeping, it is not very explicit about the fact that it is going to reduce the dose, though it hints at a reduction by saying that the "factor of reduction" is being set to 0.67.

A serious problem, and the primary one we will address in this paper is that the system *cannot* give justifications for its actions. That is, while this way of giving explanations can state *what* a system does or did, it cannot state *why* it did what it did. Justifications are an important part of explanation. They can reveal either that a system is based on sound principles or, equally importantly, show that a program is being pushed beyond the bounds of its expertise. In the explanations given above, the system cannot state that it reduces the dose because increased calcium causes increased automaticity. The information needed to justify the program is the information that was used by the programmer to write the program, but it does not have to be incorporated into the program for the program to perform successfully. Since it is desirable for expert programs to be able to justify what they do as well as do it, we need to find a way of capturing the knowledge and decisions that went into writing the program in the first place.

It is interesting to note that work in computer aided instruction has followed a similar path. Initially, canned text responses were employed, but these were found to be too constraining. Later attempts were made to teach the rules used by expert systems [Carr77b,Clancey79]. Clancey [Clancey79] notes that these rule-based systems presented some explanation problems verv similar to the ones described here. More advanced work recognizes that for successful teaching, the problem domain and problem solving mechanisms must be explicitly represented [Brown75, Clancey81].

5. Providing Justifications

We need a way of capturing the knowledge and decisions that went into writing the program. One way to do this is to give the computer enough knowledge so that it can write the program itself and remember what it did. While automatic programming itself has been researched considerably [Balzer77, Barstow77, Green79, Long77, Manna77], the idea of using an automatic programmer to help in producing explanations is new. Since we are primarily interested in explanation, we have chosen not to deal with a number of problems that arise in automatic programming, including: choosing between different implementations, backup and recovery from dead-end refinements, and optimization.

XPLAIN's automatic program writer creates augmented performance programs which not only perform the intended task, but also can be explained and justified. Figure 4 illustrates some of the kinds of explanations the system can produce. Note that the system can justify its actions in terms of causal relations in the medical domain and that it can suggest analogies with previous explanations. These explanations should be compared with those in Figures 1-3 to appreciate the improvement.

An overview of the XPLAIN system is given in Figure 5. The system has five parts: a Writer, a domain model, a set of domain principles, an English Generator, and a generated refinement structure. The Writer is an automatic programmer. It has been used to write new code which captures the functionality of major portions of the old Digitalis Advisor. The domain model and the domain principles contain knowledge about the domain of expertise, in this case, digitalis and digitalis therapy. They provide the Writer with the knowledge it needs to write the code for the Digitalis Advisor. The refinement structure can be thought of as a trace left behind by the Writer. It shows how the Writer develops the Digitalis Advisor. When a physician user runs the Digitalis Advisor, he can ask the system to justify why the program is doing what it is doing. The Generator gives him an answer by examining the refinement structure and the step of the Advisor currently being executed. If we wanted to write a new program covering a new medical domain, we would have to change the domain model and the domain principles, but we should not have to change the Writer or the English Generator. (The refinement structure would be generated by the Writer.)

Fig. 4. Justifications from a Program Created with XPLAIN

Please enter the value of serum potassium: why?

The system is anticipating digitalis toxicity. Decreased serum potassium causes increased automaticity, which may cause a change to ventricular fibrillation. Increased digitalis also causes increased automaticity. Thus, if the system observes decreased serum potassium, it reduces the dose of digitalis due to decreased serum potassium.

Please enter the value of serum potassium: 3.7

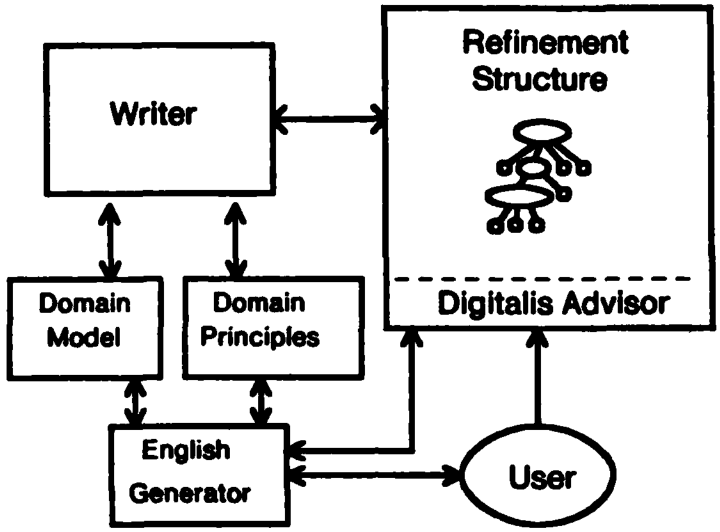
Please enter the value of serum calcium: why?

(The system produces a shortened explanation, reflecting the fact that it has already explained several of the causal relationships in the previous explanation. Also, since the system remembers that it has already told the user about serum potassium, it suggests the analogy between the two here.)

The system is anticipating digitalis toxicity. Increased serum calcium also causes increased automaticity. Thus, (as with decreased serum potassium) if the system observes increased serum calcium, it reduces the dose of digitalis due to increased serum calcium.

Please enter the value of serum calcium: 9

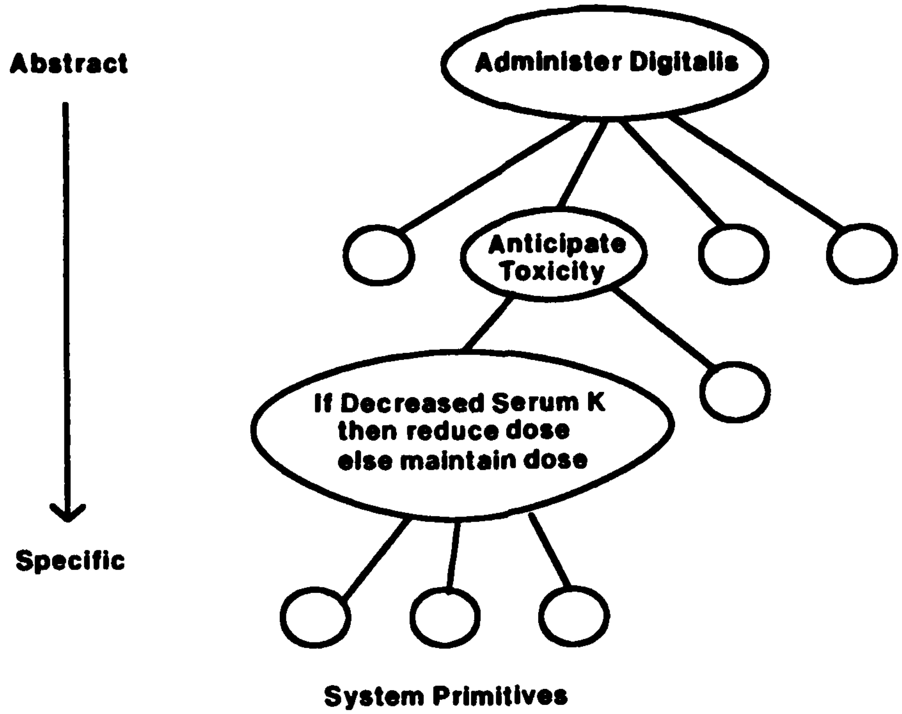
Fig. 5. System Overview



5.1 The Refinement Structure

The refinement structure is created by the Writer from the top level goal (in this case "Administer Digitalis") as it writes the Digitalis Advisor. The refinement structure is a tree of goals, each being a refinement of the one above it (see Figure 6). Here, "refining a goal" means turning it into more specific subgoals. As Figure 6 shows, the top of the tree is a very abstract goal: in this case, "Administer Digitalis". This goal is refined into less abstract steps by the Writer. These more specific steps are steps the system executes to administer digitalis. For example, one such subgoal is to "Anticipate Toxicity", that is, to anticipate whether the patient may become toxic due to increased digitalis sensitivity. The Writer then refines this more specific goal to a still more specific goal. Eventually, the level of system primitives is reached. System primitives are built-in operations. Normally they are very basic, simple operations, so the fact that they cannot be explained is usually not a problem. Typical primitives include arithmetic operations like PLUS and TIMES and those that set variables to a particular value. The leaves of the refinement structure constitute the basic operations performed by the Digitalis Advisor, the program that we wanted the automatic programmer to produce.

Fig. 6. A Sample Refinement Structure



5.2 The Domain Model

The domain model represents the facts of the domain. In this case, it is a model of the causal relationships important in digitalis therapy. The model is similar in character to the causal networks of CASNET [Weiss78], with two differences. First, the causal links are not numerically weighted. Weights could easily be added, but for this domain, most of the reasoning does not require them.[3] Second, inter-relationships between causal paths which impinge on the same state are explicitly represented.

A simplified portion of the model is shown in Figure 7. The boxes are states and the arrows represent causality. This model shows some of the effects of increased digitalis. It also shows that increased serum Ca and decreased serum K each can cause increased automaticity. These facts correspond to the sorts of facts that a medical student learns in class during the first two years of medical school. They are descriptive. While they tell what happens in the domain, they do not indicate what the Advisor's behavior should be to

3. When numerical weighting is required, XPLAIN asks the system developer for the appropriate constants during creation of the performance program.

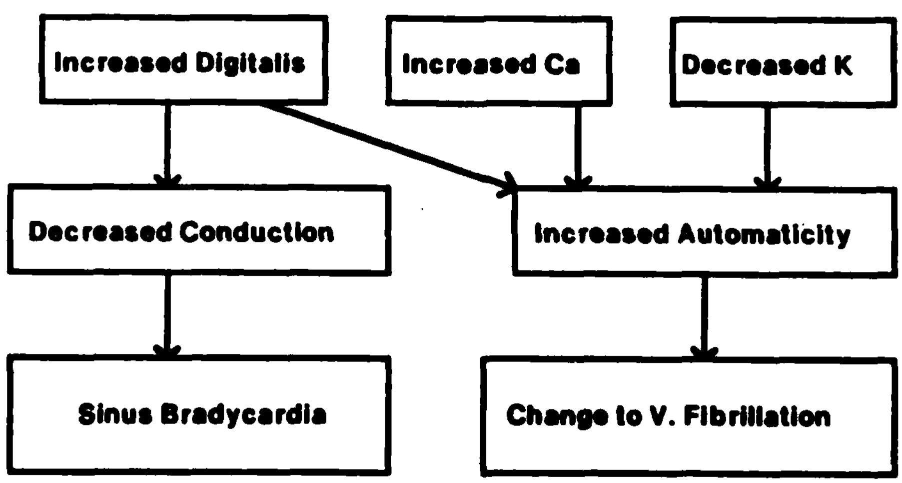
achieve its goal of administering digitalis. For example, the model says that increased digitalis can cause a change from a normal heart rhythm to ventricular fibrillation and it notes that that is dangerous, but it does not say what should be done to prevent such a change. Medical students go to medical school for an additional two years and acquire these procedures by observing experienced practitioners at work. The set of domain principles provides the Writer with this sort of procedural knowledge.

5.3 Domain Principles

Domain principles tell the Writer how something (such as prescribing a drug or analyzing symptoms) should be done. They can be thought of as abstract procedural schema which are filled in with facts from the domain model to yield specific procedures. A (somewhat simplified) domain principle appears in Figure 8. Domain principles are composed of variables and constants. Variables appear in boldface in Figure 8. Pattern variable matching uses the kind hierarchy imposed by XLMS, the knowledge representation language used by XPLAIN. For an object to match a pattern variable, it must be at the same level as or beneath the pattern variable in the hierarchy. Thus, the variable **finding** matches increased serum calcium or decreased serum potassium, since increased serum calcium and decreased K are both kinds of findings. The principle appearing in Figure 8 is used to anticipate digitalis toxicity. It represents the common sense notion that if one is considering administering a drug, and there is some factor that enhances the deleterious effects of that drug, then if that factor is present in the patient, less drug should be given. This principle has three parts: a *goal*, a *domain rationale*, and a *prototype method*. In general, a principle

---

Fig. 7. A Simplified Portion of the Domain Model



may also have a set of constraints associated with it which must be satisfied if the principle is to be used.

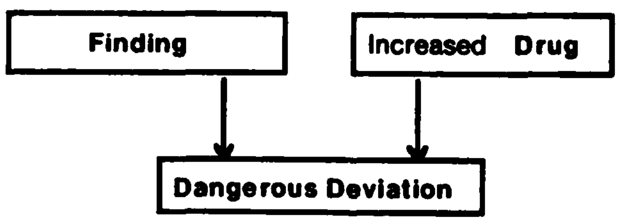
The goal tells the Writer what the principle can accomplish. In this case, the principle can help the Writer in anticipating toxicity. Domain principles are organized into a hierarchy based on the specificity of their goals. The Writer selects the principle whose goal matches the step to be refined and whose constraints (if any) are satisfied.[4] The domain rationale is a pattern which is matched against the domain model. It provides, in the context of a particular method for achieving a goal, additional information necessary for achieving the goal. In this example, the domain rationale describes which findings should be checked in anticipating toxicity. Essentially, this domain rationale defines the characteristics that a finding must have if it is to be considered when checking for digitalis sensitivities. The system looks in the domain model for every **dangerous deviation** (e.g. change to ventricular fibrillation) caused by the combination of a a **finding** (e.g. increased Ca) and an

---

Fig. 8. An Example of a Domain Principle

Goal: Anticipate **Drug** Toxicity

Domain Rationale:



Prototype Method:

If the **Finding** exists
then: reduce the **drug** dose
else: maintain the **drug** dose

---

4. If more than one principle matches the most specific one is selected. Thus, the Writer can handle special cases cleanly. For example, a general method for obtaining the value of a variable x from the user would be to ask "What is the value of x?". If the variable were boolean, a more appropriate question would be to ask "Is x present?" and expect a yes or no answer. The goal for the first method would be "Determine the value of a variable", while the goal for the second would be the more specific "Determine the value of a boolean variable".

increased **drug** level. It finds two matches: a change to ventricular fibrillation can be caused by an increased level of digitalis when combined with either increased calcium or decreased potassium.

The prototype method is an abstract method which tells how to accomplish the goal. Once the domain rationale has been matched, the prototype method is instantiated for each match of the domain rationale. Instantiating the prototype method means creating a new structure by replacing the variables in the prototype method with the things they matched. In this case, two structures are created. In the first, **finding** is replaced by increased serum Ca and **drug** is replaced by digitalis. In the second, **finding** is replaced by decreased serum K and **drug** is again replaced by digitalis. Note that these new structures, change the single abstract problem of how to anticipate toxicity into several more specific ones, such as how to determine whether increased serum K exists, how to reduce the dose, and how to maintain it.

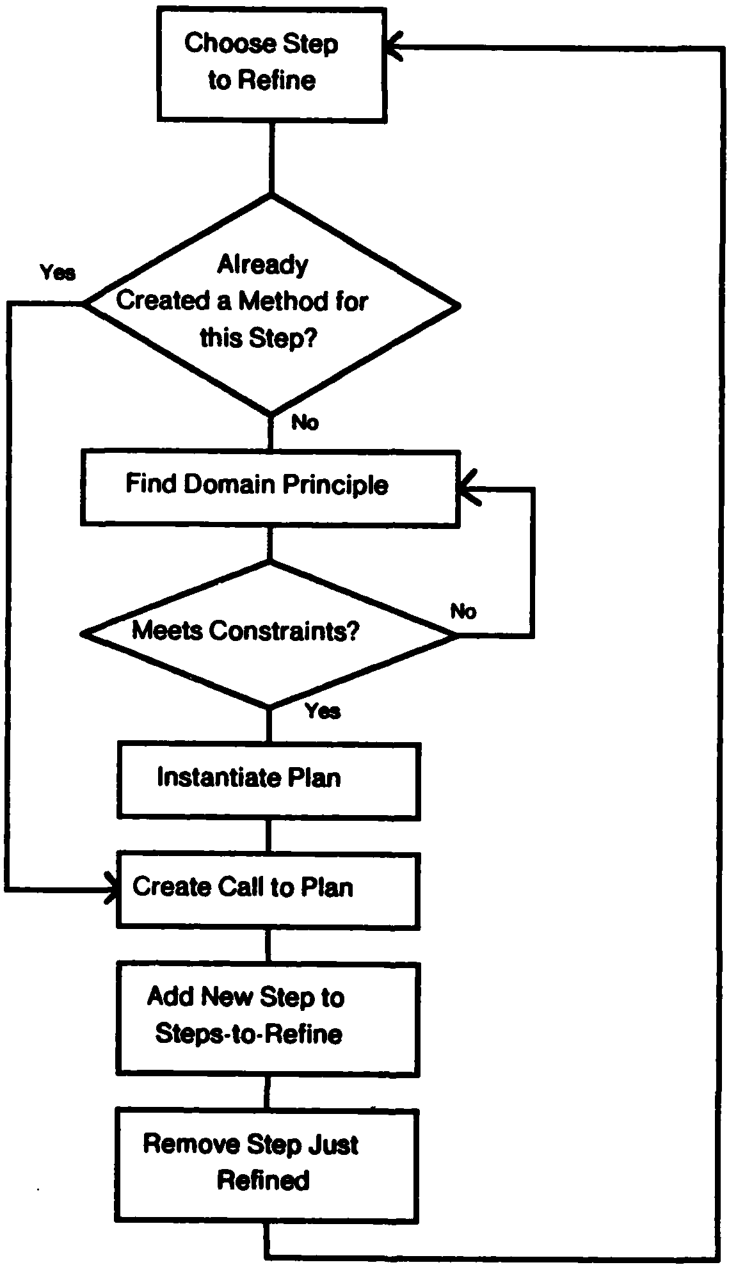
After instantiation, the goals of the prototype method are placed in the refinement structure as sons of the goal being resolved. If we look at Figure 6, we can see that the instantiated prototype method that checks for decreased serum K has been placed below the Anticipate Toxicity goal. Once they have been placed in the refinement structure, the newly instantiated goals become goals for the writer to resolve. For example, after the Writer applied this domain principle, it would have to find ways of determining whether increased calcium existed in the patient, whether decreased potassium existed, and ways of reducing and maintaining the dose. The system continues in this fashion, refining goals at the bottom of the structure and growing the tree down and down until eventually the level of system primitives is reached.

5.4 The Domain Rationale: More Detail

As was mentioned above, the domain rationale is a pattern which is matched against the domain model. It serves two purposes. First, it is a constraint on the acceptability of the domain principle, because if no matches are found, the domain principle is rejected. Second, it can also be thought of as a further specification of the performance program.

To place further restrictions on the match that would be difficult to express as a pattern, the domain rationale can have predicates associated with it which filter out unacceptable matches. The domain rationale we have been using as an example has three predicates. (These are not shown in Figure 8.) The first predicate specifies that the **finding** cannot be the increased drug level. We cannot allow that match because we are looking for *other* factors which increase the danger of giving the drug. The second predicate fails if the current value of **finding** is the same as the value it had on a previous match. This requirement ensures that for each successful match, the value of **finding** will be different

Fig. 9. Overview of the Program Writer



from its value in all other successful matches. This predicate is necessary because a particular finding may cause more than one dangerous deviation. For the purposes of this principle it is sufficient that it cause one dangerous deviation. The third predicate requires that the causal effects of the increased drug level and **finding** must be at least causally additive. This predicate checks to see whether the causal links in the causal chains between **finding** and **dangerous deviation** and between **increased drug** and **dangerous deviation** are at least additive[5] at the node where the two chains merge together. That is, the combined dangerous deviation caused by the finding together with the increased drug must be at least as great as the sum of their individual effects.

When the domain rationale is matched against the digitalis domain model, there are two matches, corresponding to the two sensitivities that are described above—increased serum calcium and decreased serum potassium. When the pattern matcher returns the matches, the matched structure and variable bindings are saved for use in producing explanations. Once all the matches have been obtained, the Writer instantiates the prototype method.

5.5 Intertwining Specification and Implementation

The notion of the domain rationale as a partial program specification is something that seems to be unique to the XPLAIN system. Generally, in other automatic programming systems, the complete specifications for the program must be given before the implementation of the program begins. Here, the specifications for the program are elaborated by matching the domain rationale against the domain model as the refinement of the program progresses. Thus, the nature of the specification will be affected by which particular domain principles are chosen for the implementation so that the process of elaborating the specification of the performance program is intertwined with its implementation [Swartout82].

This approach permits considerable flexibility and generality. The creator of the domain model has to encode only the descriptive knowledge of the domain. He does not have to worry about how that knowledge will be used in the creation of a program or worry about what that program should do (as he might if he were trying to create program specifications). New information can be added to this model and incorporated into a new version of the performance program by re-running the automatic programmer. A particular piece of knowledge might be usec for several purposes (or not at all). For example, information about the effects of increased digitalis levels is used by the system both in anticipating toxicity and in assessing toxic and therapeutic reactions.

---

5. Effects which are synergistic would be considered more than additive.

Davis [Davis76] introduced the notion of meta-rules, which bear some similarity to domain principles. However, meta-rules were used only for ordering and pruning the application of lower level rules within the context of a standard rule interpreter. Domain principles can control program refinement to a much greater degree.

The domain rationale is one of the mechanisms used in the XPLAIN system for tying the independent domain model into the specification of the performance program. Yet, the domain rationales themselves can be quite general, and are really independent of the particular domain model. The domain principle used by the system for anticipating digitalis toxicity could be used with different domain models to accomplish the same task for a number of other drugs.

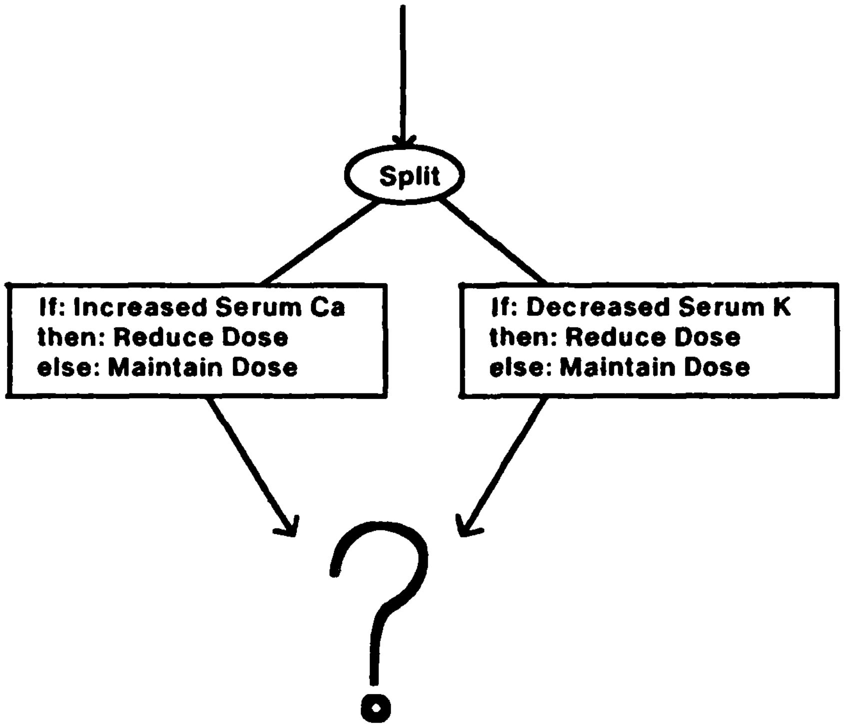
5.6 Integrating Program Fragments

The Writer must also take into account interactions between the actions it takes. For example, the individual instantiations above indicate that if increased serum calcium exists the dose should be reduced, and if decreased serum potassium exists the dose should be reduced, but they do not tell what should happen if *both* increased calcium *and* decreased serum potassium co-occur. Thus, the Writer is confronted with the problem of integrating these individual instantiations into a whole. (See Figure 10.)

Exactly what should happen depends on the characteristics of the domain. It could be that the occurrence of either sensitivity "covers" for the other, so that only one reduction should be made and the predicate of the IF should be made into a disjunction. Or, (as is actually the case), it could be that when multiple sensitivities appear, multiple reductions should be made.

Interaction problems of this sort are handled by setting them up as explicit goals to be refined. For each of the matches of the domain rationale, the Writer instantiates the prototype method. It then places each of these instantiations into a larger structure called a *split-join*. The split-join is placed in the refinement structure and domain principles are used to transform it into executable program structure. If only one match is found for the domain rationale, or if there is no domain rationale at all in the domain principle, the Writer just instantiates the prototype method and does not make up a split-join.

Fig. 10. A Split to be Resolved



Note: To keep the figure simple, only 2 sensitivities are shown

---

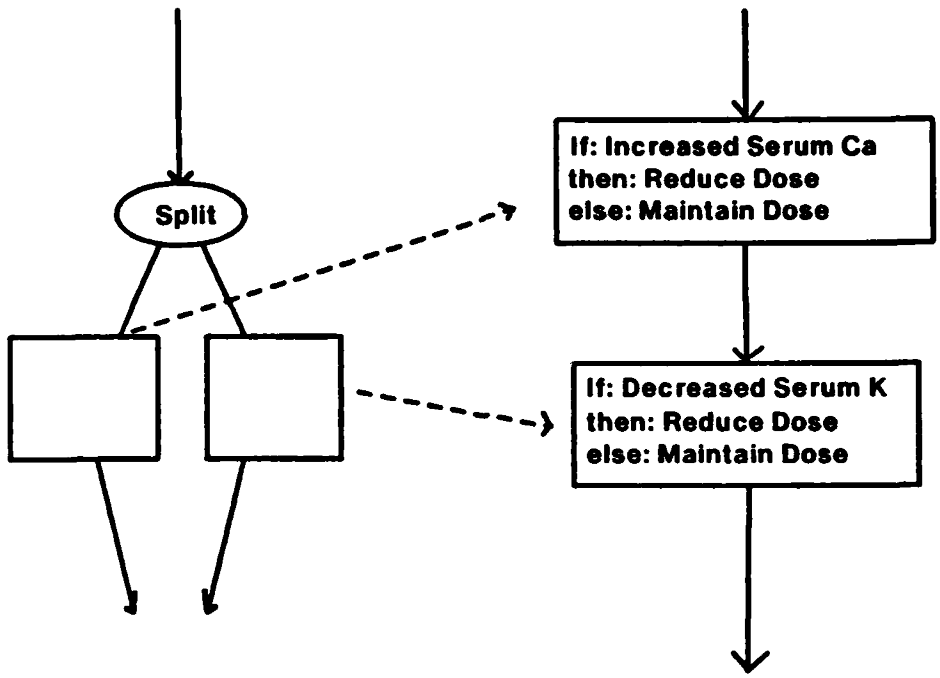
5.6.1 Refining a Split-join

The system chooses to refine the split-join next because it will result in a transformation of the program structure. It is necessary to refine transformations first because they may impose constraints on the way other steps will be refined.

The domain principle that will be used produces an executable piece of code by serializing the parts of the split join (see Figure 11). That is, the checks for increased serum calcium and decreased serum potassium will be performed in turn and the outputs for the first reduction will be connected to the second, so that if multiple sensitivities exist, multiple reductions will be performed.

The goal of the domain principle matches an arbitrary number of program fragments of the form:

Fig. 11. Resolving a Split By Serialization
<= =<dsplitrespaper.press<



Note: To keep the figure simple, only 2 sensitivities are shown

---

If Exists(**finding**)
then **adjust1**
else **adjust2**

(Note: boldface represents pattern variables)

Although the goal matches, there are some constraints associated with this domain principle which must be satisfied before it can be used to transform the program structure. The constraints check whether this serialization is a reasonable sort of resolution. There are two general types of constraints: *domain constraints* and *refinement constraints*.

Domain constraints are tests to see whether a domain principle is applicable given the characteristics of the domain. The first constraint checks that all the deviations (e.g. increased serum calcium, decreased serum potassium and cardiomyopathy) are causally independent in the sense that none of them causes the other. This is done by examining the domain model to see if there is a causal chain leading from any of the deviations to any other. If one finding causes another, serialization would be inappropriate because if both findings appeared a double reduction would be made when only the reduction for the underlying cause was appropriate.

The second constraint checks to see whether the effects of the causal chains are additive. That is, before making multiple reductions for multiple sensitivities, it must be the case that the occurrence of multiple deviations is worse than just one by itself. The domain model supplies this information.

Refinement constraints are the other type of constraints used in this domain principle. Like domain constraints, refinement constraints determine whether or not a particular domain principle is applicable, but if it is found to be applicable, they also constrain the way in which further refinements may be made.

In this particular case, we are resolving the split-join by serializing the reductions. Whether or not this is a valid way to proceed depends in part on how the reductions themselves are refined. If the reductions are to be performed by subtracting some quantity from the dose, then there is some possibility that the dose will eventually become negative. That, of course, does not make sense. So the principle for resolving the split-join would have to insert a step at the end which would check for a negative dose and do whatever was proper. On the other hand, if the dose is reduced by multiplying the original dose by some constant less than one to produce a new dose, then the dose can never become negative, and no check is required. To resolve the split-join now, the Writer needs to be able to constrain the resolution of the steps that perform the reduction.

The domain principle for refining the split-join specifies that the Writer must find domain principles for refining the two instances of [adjust1] and [adjust2] (which are the calls to reduce and maintain the dose respectively) so that that the method of the principle is described as a multiplicative operator. If a principle is found for each of the calls, the refinement constraint is satisfied. As the principles are found, the system remembers them by associating them with the appropriate entry in the list of steps to be refined.

5.7 Assessing Toxicity

Whenever the dosage of digitalis is being adjusted, it is necessary to monitor the patient closely to determine the effect (if any) of the change. In the old Digitalis Advisor, there were two sets of routines for assessing the drug's effects. One set was concerned with the harmful toxic effects of digitalis, while the other dealt with the therapeutic effects. Each set of routines produced an assessment of the degree to which the patient was showing either toxic or therapeutic effects. Based on these assessments, the system recommended corrective actions if needed. This section briefly describes how the portion of the Digitalis Advisor that assesses toxicity was implemented using the XPLAIN system. A more complete discussion may be found in [Swartout81].

To assess toxicity, the user is asked whether the various toxic effects that digitalis may cause have been observed in the patient. The assessments of these individual findings are then combined into an overall assessment of toxicity. The assessment is a number representing the degree of toxicity and the individual assessments are combined together using numerical techniques.

The cardiologists we consulted felt that when assessing digitalis toxicity they looked for signs in three general classes: highly specific signs of digitalis toxicity, moderately specific signs, and signs with low specificity (also called non-specific signs). The original Digitalis Advisor adopted these classifications and weighted the various findings according to their classification to produce an assessment of digitalis toxicity. To implement this algorithm, the domain model had to be augmented to indicate the various types of findings that could result from digitalis toxicity, and the specificity of those findings. The domain model used by the implementations appears in Figure 12.

It was also necessary to add a number of new domain principles. The top level principle to assess digitalis toxicity just sets up the goals to assess the highly specific signs, moderately specific signs and non-specific signs and to then combine them together. The three different assessment steps are all refined by the same domain principle because that principle has the degree of specificity as a variable in its goal:[6]

Assess **specific** findings of **drug** toxicity.

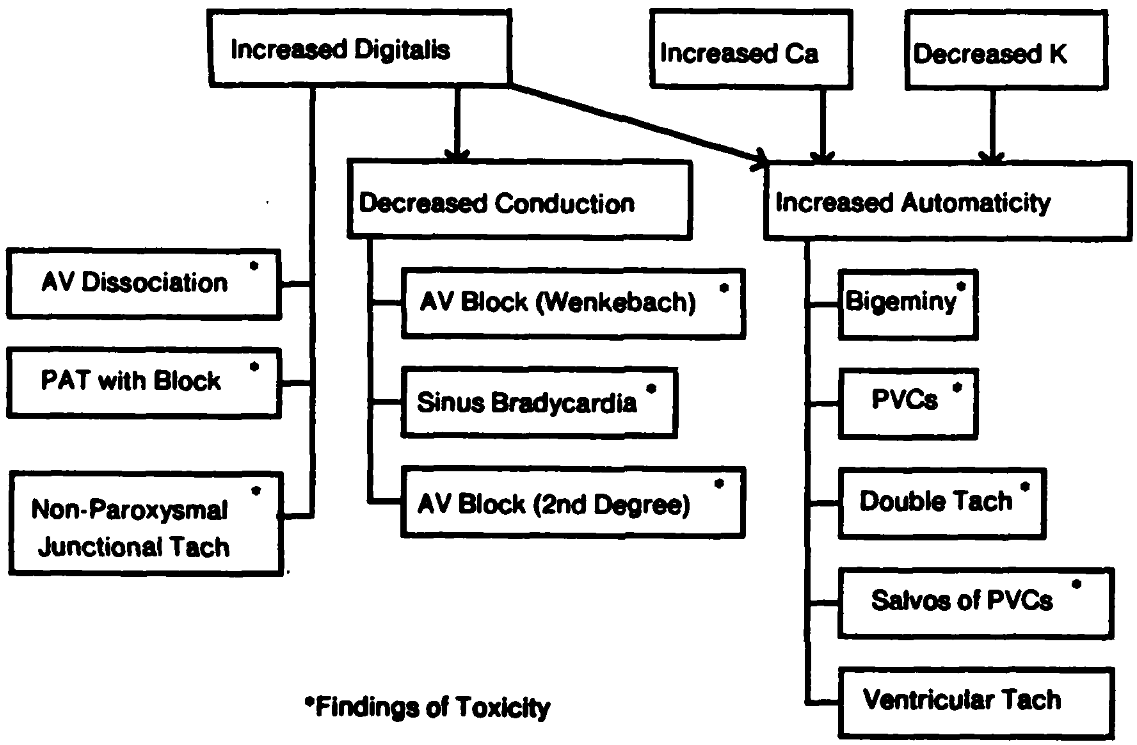
The domain rationale looks for all findings caused by increased digitalis that have the degree of specificity specified in the call. For example, when the call is "Assess highly specific findings of digitalis toxicity" the domain rationale will find paroxysmal atrial tachycardia with block, double tachycardia, and av dissociation, because these are all highly specific signs of digitalis toxicity.

The prototype method then sets up calls to assess these various findings. For most of the findings, the system just asks the user whether the finding is present or not. Some findings, such as premature ventricular contractions (PVCs), are properly assessed by comparing their current level to a patient-specific baseline. A special purpose domain principle is provided for assessing PVCs. Since the Writer always picks the most specific

---

6. This is the English translation of the goal used in the system. Again, variables appear in boldface. Here, **specific** is a variable which can take on the values *highly specific, moderately specific*, or *non-specific*.

Fig. 12. Domain Model For Toxicity



---

principle for the call being refined, this special-case knowledge is automatically employed at the appropriate time. The code for assessing findings is not normally explained to physicians, because it is unlikely to interest them. The viewpoint mechanism (described below) is used to encode that fact.

It is encouraging to note that little additional mechanism needs to be added to deal with assessing toxicity, a problem which seems quite different from the earlier examples. The algorithm of the program writer was not altered, and much of the domain model needed for dealing with sensitivities was applicable in assessing toxicity.

6. Generating Explanations

We have described how XPLAIN creates the refinement structure and the characteristics of the domain principles and the domain model. These are used by the explanation generator to provide English justifications of the program's behavior and descriptions of what the program does and how it does it.

This section describes how XPLAIN produces explanations. By design, the knowledge structures left behind by the automatic programmer make it possible to achieve quite high quality English output with a simple generator. The generator is an engineering effort aimed at producing acceptable English. The main thrust of the work described in this paper has been to investigate ways of representing the knowledge necessary to justify the behavior of expert consulting systems. A generator is necessary to demonstrate the capabilities of the approach espoused here, but is not the focus of the research. (See [Mann80, McDonald80] for a discussion of current work in generation.)

The generator is really composed of two types of generators. The low level *phrase generator* constructs phrases directly from XPLAIN's knowledge base. Higher level *answer generators* determine what to say—they select the parts of the knowledge representation to be translated into English by the phrase generator in response to specific questions. The answer generators must deal with the issue of determining at what level an explanation should be given. In making this determination, it employs knowledge of the state of the program execution, knowledge of what has already been said, and knowledge of what the user is likely to be interested in. Other issues the answer generators confront include deciding whether to omit information the user can be presumed to know from the explanation and determining whether analogies can be suggested to previous explanations. The phrase generator and the knowledge representation language used by XPLAIN, called XLMS, will be briefly discussed in the next section, followed by a discussion of the answer generators. For a more detailed discussion, see [Swartout81].

6.1 XLMS Notation and the Phrase Generator

The XPLAIN system uses XLMS to manage its knowledge base. XLMS (which stands for eXperimental Linguistic Memory System) was developed by Lowell Hawkinson, William Martin, Peter Szolovits and members of the Clinical Decision Making and Automatic Programming Groups at MIT. Since a complete understanding of the intricacies of XLMS is not needed to understand the XPLAIN system, this section is only intended to give the reader a brief overview of XLMS. For a more complete discussion of the design goals and implementation of XLMS see [Hawkinson80, Martin79].

For the purposes of this paper, perhaps the best way to think of XLMS is that it is an extension of LISP that allows one to use *structured names*. In LISP, atoms are used to name variables and functions. In the XPLAIN system, variables and procedures are named by XLMS concepts—the difference is that these concepts can have a substructure which can be taken apart and examined, while LISP atoms are indivisible.

6.1.1 XLMS Concepts

In XLMS, every concept is composed of an ilk, a tie and a cue and is written as:

```
[(<ilk>*<tie> <cue>)]
```

or, to pick an actual example from the XPLAIN system:

```
[(LEVEL*R DIGITALIS)]
```

The *ilk* of a concept is itself a concept. It tells what kind of a concept this is. Thus, the example concept is a kind of level. The *cue* of a concept is either a concept or a LISP atomic symbol. It indicates what it is that makes this concept different from others with the same ilk. The example represents the "level of digitalis": a particular kind of level. Finally, the *tie* of a concept indicates the relationship between the ilk and the cue. In this case, the tie is R for "role". Role ties are used to indicate slots in concepts. Thus this concept represents the "level" slot in the concept "digitalis". This is one implementation of the notion of slots and frames as described by Minsky [Minsky75]. There are several ties that are used extensively in the XPLAIN system. Some of these are listed in Table I together with examples of their use.

Concepts in XLMS are organized into a kind hierarchy. The root concept is [summum-genus] (see Figure 13) and is pre-defined in XLMS. *s ties create a strict taxonomy of mutually exclusive subclasses. Like atomic symbols in LISP, concepts in XLMS are unique.

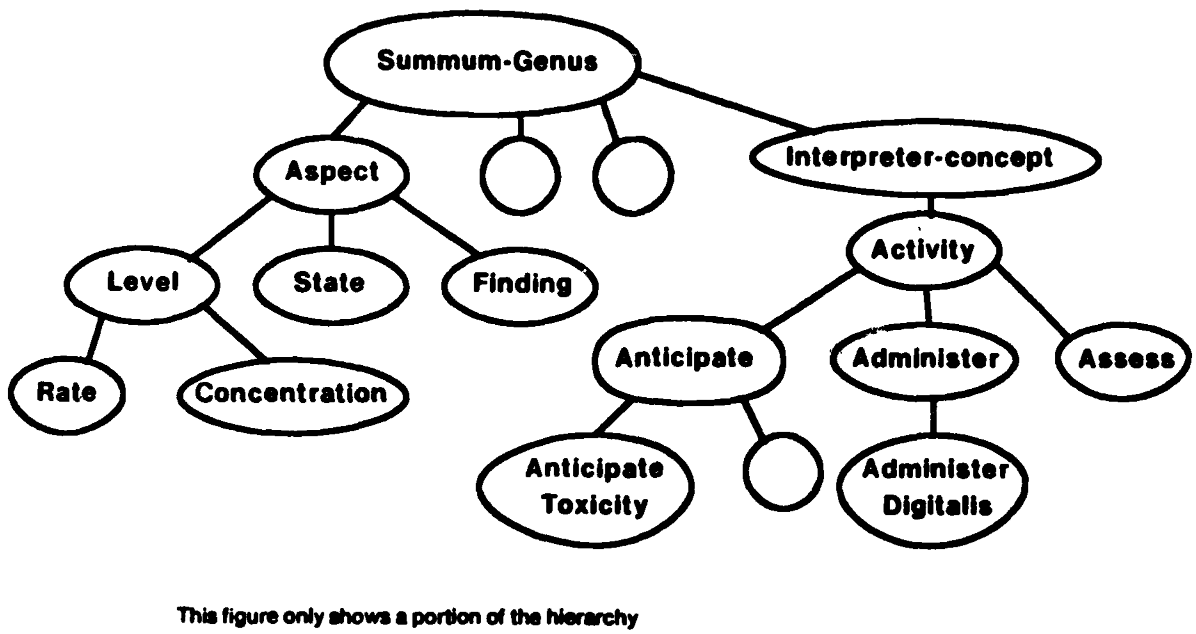
In LISP, it is possible to associate lists and atoms relating to a particular atomic symbol with that symbol by placing them on that symbol's property list. In XLMS, one can associate concepts relating to a concept, with that concept, by *attaching* them. The viewpoint mechanism (described more fully below) uses attachments placed on concepts by the Writer to determine whether or not the concept should be explained.

In the XPLAIN system, a phrase generator is associated with each kind of tie. To generate English for a particular concept, the tie of the concept is examined and control is passed to the corresponding phrase generator. The structured nature of XLMS concepts

Table I. Types of Ties (partial list)

Tie	Name	Example Use	English Form	Purpose
*f	function	`[(ball*f red)]`	(the) red ball	modifies
*r	role-in	`[(color*r ball)]`	(the) color of (the) ball	slot-filling
*i	individual	`[(ball*i 1)]`	ball	instantiates individual
*o	object	`[(treat*o patient)]`	treat (the) patient	verb-object
*s	species	`[dog = (animal*s "dog")]`	dog	(see text)

Fig. 13. The Kind Hierarchy



---

and the fact that XLMS is a linguistically oriented representation language makes it relatively straightforward to construct a phrase generator that can turn an XLMS concept such as:

```
[(assess*o (level*r digitalis))]
```

into the English phrase:

Assess the level of digitalis

The phrase generators are completely described in [Swartout81]. The next section discusses the more interesting question of how XPLAIN chooses what to say.

6.2 The Answer Generators: Determining What to Say

6.2.1 Viewpoints

The reader may recall that one problem with previous explanation systems has been the problem of *computer artifacts*. Computer artifacts are parts of the program which appear mainly because we are implementing an algorithm on a computer. If these steps are described to physicians, they are likely to be uninteresting and potentially confusing. The introductory section gave some examples of these computer artifacts. In the XPLAIN system, we can attach viewpoints to steps in prototype methods to indicate to whom the step should be explained.[7] When a prototype method is instantiated, the instantiated steps will share these viewpoints. As the XPLAIN system is generating an explanation for a step it compares the viewpoint(s) of the step (if any) against a list of viewpoints which should be filtered out and another list of viewpoints which should be included. If one of the step's viewpoints appears on the include list, that step is included in the English explanation. If not, and one of the viewpoints appears on the exclude list, the step is excluded from the explanation. If the step has no viewpoint, it is included in the explanation. This approach allows us to separate those steps that are appropriate for a particular audience from those that are not. Of course, the exclude and include lists may be changed to reflect a change in the user's viewpoint.

While this is a simple solution from the standpoint of generation, it is a feasible one because we are employing an automatic programmer. In the domain principles, we bring together and define for the system to use, computer implementation knowledge and medical reasoning knowledge. A domain principle is thus the appropriate place to indicate what viewpoint should be taken on the knowledge that it is composed of. By placing a viewpoint

7. This can occur either during the refinement of a step from a higher goal, or during a transformational refinement.

on a step in a prototype method, we cause all the instantiations of that step (and there are usually several) to share that viewpoint. If we were to try to do the same thing at the level of the performance program (without an automatic programmer) we would have to annotate each individual step—we could not capture as high a level of abstraction.

This result is consistent with the observation we made in the introduction: improvements in the quality of the explanations generated resulted more from the use of an automatic programmer than from increases in the sophistication of the generation routines. It should be pointed out, however, that while this solution allows the system to customize its explanations based on a particular viewpoint or set of viewpoints, the problem of deciding which viewpoint to present to a particular user remains open and is beyond the scope of this paper.

6.2.2 Answering "Why" Questions

One of the chief goals in this research was to have the XPLAIN system explain why the performance program was doing what it was doing. In producing answers to such questions, the system makes use of the knowledge in the domain model and the domain principles as well as traces left behind by the automatic programmer resulting from creation of the performance program. Some of the capabilities of the system were illustrated in the section 5 where the performance program was anticipating digitalis toxicity. Additional capabilities are illustrated below from those portions of the advisor that anticipate toxicity and assess toxicity.

Is the patient showing signs of cardiomyopathy? (yes or no): why?

The system is anticipating digitalis toxicity. Cardiomyopathy causes increased automaticity, which may cause a change to ventricular fibrillation. Increased digitalis also causes increased automaticity. Thus, if the system determines that cardiomyopathy exists, it reduces the dose of digitalis due to cardiomyopathy.

This explanation is similar to those in Section 5.

The remaining explanations are produced while assessing toxicity.

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no): why?

The system is assessing the highly specific findings of digitalis toxicity. Increased digitalis may cause paroxysmal atrial tachycardia with block which is a highly specific finding of digitalis toxicity.

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no):
overview

The system repeats the question, but the user wants an overview. This is produced by describing the method for assessing highly specific findings which was mentioned in the previous explanation.

To assess the highly specific findings of digitalis toxicity:

1. **The system assesses paroxysmal atrial tachycardia with block.**
2. **It assesses double tachycardia.**
3. **It assesses av-dissociation.**
4. **It combines the assessments of paroxysmal atrial tachycardia with block, double tachycardia and av-dissociation.**

This produces the assessment of the highly specific findings of digitalis toxicity, which is used when the system combines the assessments of the highly specific findings of digitalis toxicity, the moderately specific findings of digitalis toxicity and the non-specific findings of digitalis toxicity.

To avoid leaving the user hanging, the system describes how the output of the method will be used in a higher context. This is done whenever the output of the method is something which is not likely to be familiar to the user. Thus, if the output of a method were a drug dose, no additional explanation would be generated (at least for a medical audience) because the user would be presumed to be familiar with the notion of a drug dose.

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no):
why?

Still curious, the user asks "why?" again. This causes the system to give an explanation of the procedure that calls the procedure for assessing the highly specific findings. This explanation is much shorter, because the domain principle used to refine the higher level procedure had no domain rationale.

The system is assessing digitalis toxicity. One step in doing that is to assess the highly specific findings of digitalis toxicity.

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no):
overview

This time the description is offered at the level of the higher procedure.

To assess digitalis toxicity:

1. **The system assesses the highly specific findings of digitalis toxicity.**
2. **It assesses the moderately specific findings of digitalis toxicity.**

3. It assesses the non-specific findings of digitalis toxicity.
4. It combines the assessments of the highly specific findings of digitalis toxicity, the moderately specific findings of digitalis toxicity and the non-specific findings of digitalis toxicity.

This produces the assessment of digitalis toxicity, which is used when the system adjusts the dose of digitalis.

When a "why" question is entered, control passes to the routine that produces justifications. This routine determines at what level the description should be given, states what is going on in general, describes the domain rationale (if any) used in refining the step being described, and finally describes the step. Each of these four stages will be outlined below.

6.2.2.1 Choosing the Level of Description

The system uses the viewpoint attachments to determine where to start the explanation. The control stack of the interpreter executing the performance program is available to the explanation modules. The justification routine goes up the stack looking for the first procedure which is not an excluded viewpoint and which has no procedure that has an excluded viewpoint above it. If that procedure happens to be a system primitive[8] with a system primitive above it, then the system keeps going up the stack until it finds a procedure which does not have a system primitive above it.

Consider the situation presented in the sample interaction above where the system first queries the user about the "signs of paroxysmal atrial tachycardia" and the user responds "why?". As was mentioned earlier, the procedures for assessing findings, like paroxysmal atrial tachycardia, have computer-viewpoints, indicating that physicians will not be interested in them. Therefore, the system will start giving its explanation at the next level up, at the level of the call to that procedure. This will be referred to as the *current description level.* In contrast, if the exclude list had not contained computer-viewpoint, explanation would have begun at a lower level, producing the following explanation:

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no):
why?

The system is assessing paroxysmal atrial tachycardia with block. If the status of

8. System primitives include assignment statements, conditionals, arithmetic operators and the like.

paroxysmal atrial tachycardia with block is equal to present, the assessment of paroxysmal atrial tachycardia with block is set to the assessment level for present findings (1), otherwise the assessment of paroxysmal atrial tachycardia with block is set to the assessment level for absent findings (0).

This is the sort of information a person maintaining the advisor might wish to know, but that a medical audience certainly would not.

6.2.2.2 Stating What's Going On in General

To give the user an overview of what the system is trying to accomplish, the system finds the next procedure above the current description level in the control stack. This will be called the *higher level procedure*. It then generates a phrase using the name of the procedure to describe what is going on:

The system is assessing the highly specific findings of digitalis toxicity.

6.2.2.3 Explaining the Domain Rationale

If the higher level procedure was refined using a domain principle which had a domain rationale, then the procedure at the current description level must be the result of one of the matches of the domain rationale. The system finds the domain rationale and the particular match of it that resulted in the procedure at the current description level. Flags are set to indicate to the phrase generator that it should replace occurrences of pattern variables with the objects they matched. After this environment has been set up, the complete pattern is found and converted to English using the phrase generator. For example, the phrase generator produced the following description of the domain rationale of the domain principle that refined the procedure to assess highly specific toxic findings:

Increased digitalis may cause paroxysmal atrial tachycardia with block which is a highly specific finding of digitalis toxicity.

6.2.2.4 Finishing Up the Explanation

Finally, the system uses the phrase generator to produce a description of the step at the current level of description. So when the user replies "why?" to the program's question about cardiomyopathy, XPLAIN generates the following description of the current step:

Thus, if the system determines that cardiomyopathy exists, it reduces the dose of

digitalis due to cardiomyopathy.

The system then re-iterates its original question. If the user asks "why?" again, the system moves the current description level up to the level of the next higher level procedure and repeats the explanation process.

Sometimes this description is omitted. When the user types "why?" in response to the program's first question about paroxysmal atrial tachycardia, the current level of description is "assess paroxysmal atrial tachycardia with block". Yet, XPLAIN does not produce the sentence:

"Thus, the system assesses paroxysmal atrial tachycardia with block."

as the last sentence of its answer to the second question. The reason is that such a sentence would have been redundant. The user already knows that the system is assessing paroxysmal atrial tachycardia with block, because he has just been asked a question about it. Following the general principle that the user should not be told something he already knows, the system deletes this part of the explanation if the step about to be described is a type of assessing step and the object of that step is the same as the thing the user has been asked about.

Given the system described so far, it is relatively easy to get it to generate descriptions of methods by translating them directly into English. The explanations given by the "overview" command in the previous section were produced by passing the current higher level procedure to the function that describes methods. However, as we pointed out in the introduction, this particular style of explanation has some limitations. In the next section, we present a different way of explaining methods which provides a richer sort of abstraction which cannot be done in explanations produced directly from the code.

6.2.3 Domain Principle Explanations

In the original version of the Digitalis Advisor, when we wanted to give a more abstract view of what was going on, we just described a higher level procedure [Swartout77a, Swartout77b]. In this regard, we were following the principles of structured programming. While this approach was often reasonable, there were times when it was considerably less than illuminating. The general method for anticipating digitalis toxicity was called "Check Sensitivities" in the old version of the Digitalis Advisor. An explanation of it appears in Figure 14. While this explanation does tell the user what sensitivities are being

Fig. 14. An Explanation From the Old Digitalis Therapy Advisor

(describe-method [(check sensitivities)])

```
TO CHECK SENSITIVITIES I DO THE FOLLOWING STEPS:
1. I CHECK SENSITIVITY DUE TO CALCIUM.
2. I CHECK SENSITIVITY DUE TO POTASSIUM.
3. I CHECK SENSITIVITY DUE TO CARDIOMYOPATHY-MI.
4. I CHECK SENSITIVITY DUE TO HYPOXEMIA.
5. I CHECK SENSITIVITY DUE TO THYROID-FUNCTION.
6. I CHECK SENSITIVITY DUE TO ADVANCED AGE.
7. I COMPUTE THE FACTOR OF ALTERATION.
```

---

checked,[9] it does not say what will be done if sensitivities are discovered nor does it say why the system considers these particular factors to be sensitivities. Finally, it is much too redundant and verbose. The first objection can be dealt with by removing the calls to lower procedures and substituting the code of those procedures in-line. This results in the somewhat improved explanation produced by XPLAIN when it is asked to describe the method for anticipating digitalis toxicity (see Figure 15). However, while this explanation shows what the system does, it does not say why things like increased calcium, cardiomyopathy and decreased potassium are sensitivities, and if anything, it is even more verbose than the original explanation.

The reason we cannot get the sorts of explanations we want by producing explanations directly from the code is that much of the sort of reasoning we want to explain has been "compiled out." Thus, we are forced into explaining at a level that is either too abstract or too specific. The intermediate reasoning which we would like to explain was done by a human programmer in the case of the old Digitalis Advisor. However, because this performance program was produced by an automatic programmer, we have a handle on that reasoning. For example, if we were to explain the domain principle that produced the code for anticipating digitalis toxicity rather then the code itself we would get the explanation that appears in Figure 16. This explanation is produced by first describing the

---

9. The reader may notice that there were more sensitivities checked in the original version of the program than in the current version. We now feel that some of these, such as thyroid function and advanced age, should not be treated as sensitivities per se because they tend to have an effect on reducing renal function and hence slowing excretion, rather than on increasing sensitivity to digitalis. The other sensitivities would be easy to add by including the appropriate causal links in the domain model.

Fig. 15. An Explanation From the Code for Anticipating Toxicity

(describe-method [((ANTICIPATE*O (TOXICITY*F DIGITALIS))*I 1)])

To anticipate digitalis toxicity:

1. **If the system determines that cardiomyopathy exists, it reduces the dose of digitalis due to cardiomyopathy.**
2. **If the system determines that decreased serum potassium exists, it reduces the dose of digitalis due to decreased serum potassium.**
3. **If the system determines that increased serum calcium exists, it reduces the dose of digitalis due to increased serum calcium.**

Fig. 16. Explanation of a Domain Principle

(describe-proto-method [(anticipate*o (toxicity*f digitalis))])

The system considers those cases where a finding causes a dangerous deviation and increased digitalis also causes the dangerous deviation. If the system determines that the finding exists, it reduces the dose of digitalis due to the finding.

The findings considered are increased serum calcium, decreased serum potassium and cardiomyopathy.

---

domain rationale with the refinement pattern variables [10] replaced by what they matched, but with the domain pattern variables described as themselves rather than as what they matched. Thus while the system says "increased digitalis" rather than "increased drug", it nevertheless says "finding" rather than "increased serum potassium". The next part of the explanation is produced by describing the prototype method. Finally, the set of values is given for the domain variable used in the prototype method. Thus, the use of an automatic programmer not only allows us to justify the performance program, but it also allows us to give better descriptions of methods by making available intermediate levels of abstraction

10. Those are the variables in the head of the domain principle that were bound during plan finding by the automatic program writer.

which were not previously available.

6.2.4 The Domain Rationale: Its Role in Explanation

The above example also illustrates the role that the domain rationale plays in allowing us to provide better quality explanations. Programming in the top-down structured programming methodology can be thought of as moving between levels of language. Nodes higher in the development structure are stated in a higher level language than those that are lower down. However, as Figure 14 illustrates, the correspondences (or "translations") between those language levels are implicit. In the example, the method `[(check sensitivities)]` is at a higher level than the methods it calls which check individual sensitivities. Yet, the reasons why those sensitivities are sensitivities are not represented. The domain rationale is used to indicate those correspondences. For example, the domain rationale of the domain principle used for anticipating toxicity essentially defines what it means for a finding to be a sensitizing factor in digitalis therapy. Paraphrasing the description given in Figure 16, it is a finding that causes a dangerous deviation that is also caused by increased digitalis. It should be noted that the current implementation should be more explicit about exactly which terms in the higher level language are being redefined at a lower level by the domain rationale. This limitation could impair the quality of explanations in more complex situations than were encountered in the Digitalis Advisor. For example, if multiple terms were being re-defined, the current implementation would have trouble sorting everything out. That limitation aside, the major point here is that the domain rationale allows us to be much more explicit about how moves between levels of language are being made. This, in turn, makes it possible to provide substantially better explanations.

6.2.5 Explaining Events

The interpreter of the performance program can be set up to leave behind a trace of its execution. As it executes a procedure, it creates an *event object*, which records the call and method used, the variable environments on entrance and exit, and the value returned if the procedure is a functional subroutine. These events can be examined by the system after execution is completed to produce an explanation of what the system did for a particular patient.

Once we have the mechanisms in place to explain methods it turns out to be quite easy to explain events. Basically, it is done by having the system examine the event to be explained, generate a heading sentence using the call that caused the event, and then generate phrases for the immediate subevents of the event. As when explaining methods, the subevents are filtered by their viewpoints.

The major changes in the explanation routines that have to be made are that a flag has to be set so that verbs are generated in the past tense, and the generator for conditionals has to be modified to indicate the choice taken. This is done by first having the generator check that there was an action taken by the conditional and then having it generate English for the predicate and the action taken. For example, the conditional does not take an action if its predicate evaluated to false and there was no "else" clause, or if the action taken was of a viewpoint that was filtered out. If no action was taken, or the action taken is filtered out, the system just generates English for the predicate (for example, see steps 1 and 2 in Figure 17.) Finally, the system generates a phrase indicating the final output values of the routine.

6.2.6 Non-English Explanation

There are many situations in which English is not the only or best way to give an explanation. Many times, explanations are much more effective when English text is supplemented with figures, charts, drawings and so forth.

Fig. 17. Examples of Event Explanations

"How did the system anticipate digitalis toxicity?"

*(describe-event [(event*i "e0002")])*

To anticipate digitalis toxicity:

1. The system determined that cardiomyopathy did not exist.
2. The system determined that decreased serum potassium did not exist.
3. Since the system determined that increased serum calcium existed, the system reduced the dose of digitalis due to increased serum calcium.

The adjusted dose of digitalis was set to 0.20.

"How did the system determine that serum calcium was increased?"

*(describe-event [(event*i "e0016")])*

Since serum calcium (13) was greater than the threshold of increased serum calcium (11) the system determined that increased serum calcium existed.

Fig. 18. Describing Events with Arithmetic Expressions

"How did you reduce the dose for increased serum calcium?"

The dose after adjusting for increased serum calcium was set according to the following formula:

D2 = D1 C

where:
C = the reduction constant for increased serum calcium (0.8)
D1 = the dose before adjusting for increased serum calcium (0.25)
D2 = the dose after adjusting for increased serum calcium (0.20).

The dose of digitalis adjusted for the condition of the heart muscle, serum potassium and serum calcium was set to 0.20.

A case in point is explaining mathematical formulas. Mathematical formulas expressed in English are not only verbose; they are ambiguous as well [Swartout77a]. As a small step in moving toward a larger investigation of non-English explanations, the XPLAIN system describes arithmetic expressions using mathematical notation. This is done by choosing shortened variable names for the variables and converting the prefix XLMS form for arithmetic expressions to an infix form which is printed. Figures 18 and 19 show some examples.

7. A Discussion of the Automatic Programming Approach to Explanation

This section addresses several issues that arose while implementing the system. Most of these deal with the interrelationships between the automatic programmer, the performance program, and the explanations that can be produced.

7.1 Does Automatic Programming Affect the Performance Program?

We attempted to get the XPLAIN system to write procedures that captured the intent of the corresponding sections of the original Digitalis Advisor as much as possible. However, there were situations where we decided to adopt different strategies. Usually this occurred because the attempt to find domain principles to generate the program forced us

Fig. 19. Describing Methods with Arithmetic Expressions

"How does the system combine the assessments of highly specific, moderately specific, and non-specific findings of toxicity?"

The combined assessment of the highly specific findings of digitalis toxicity, the moderately specific findings of digitalis toxicity and the non-specific findings of digitalis toxicity is set according to the following formula:

C = F1 A1 + F2 A2 + F3 A3

where:
A1 = the assessment of the highly specific findings of digitalis toxicity
A2 = the assessment of the moderately specific findings of digitalis toxicity
A3 = the assessment of the non-specific findings of digitalis toxicity
C = the combined assessment of the highly specific findings of digitalis toxicity, the moderately specific findings of digitalis toxicity and the non-specific findings of digitalis toxicity
F1 = the weighting factor of the highly specific findings of digitalis toxicity (4)
F2 = the weighting factor of the moderately specific findings of digitalis toxicity (2)
F3 = the weighting factor of the non-specific findings of digitalis toxicity (1).

---

to look more closely at the methods we were adopting, and occasionally we discovered that the original program was flawed or inconsistent.

For example, in the original Digitalis Advisor, myxedema[11] was considered as one of the digitalis sensitivities (like increased calcium or decreased potassium). In creating the domain model and domain principle to anticipate toxicity in the new system, we realized that a problem existed because myxedema was not causally additive with the other sensitivities and hence would not meet all the refinement constraints required by the domain principles in refining the program. To resolve the problem, we dug deeper into the medical literature and discovered that myxedema should not really be considered a sensitivity at all! In fact, myxedema reduces the excretion of digitalis through the kidneys and thus tends to make

---

11. Myxedema is a disease caused by decreased thyroid function. Signs of the disease include dry skin, swellings around the lips and nose, mental deterioration, and a subnormal basal metabolic rate.

digitalis accumulate in the body rather than making the patient more sensitive to digitalis. Therefore, the appropriate way to handle myxedema is not as a sensitivity, but as a factor which modifies the excretion rate in the pharmacokinetic model.

One of the advantages of the automatic programming approach is that it forces the user to think harder about the performance program and its implementation. Just as the implementation of any theory on a computer forces one to work out the details and think about the consistency of the theory, working out the implementation of the implementation carries the process one step further and forces one to think that much harder about the entire undertaking.

7.2 Is this Approach to Explanation Compatible with Others?

The approach to explanation espoused in this thesis is compatible with other approaches such as using canned text or producing explanations by translating the program code. It should be regarded as an extension of these earlier approaches rather than a replacement for them. This is important because there may be times when it is not feasible to get an automatic programmer to produce the code. The XPLAIN system allows the user to hand code parts of the system and can generate the remainder automatically. Those parts of the system that are hand-coded can be translated to English just like the parts that are automatically generated. The current implementation of the XPLAIN system does not support canned-text explanations (mainly because they have not been needed) but could easily be modified to do so.

7.3 Is Automatic Programming Too Hard?

One possible objection to the whole approach to explanation advocated here is that it is just too difficult to get an automatic programmer to write the performance program. When I first began this research, I thought that was the case. The original plan for producing better explanations was to create structures detailing the development of the performance program, but these structures would be created by hand rather than automatically. It was feared that automatic programming was just too hard. However, as the research progressed, it became clear that if we had sufficiently powerful representations available so that it could be said that in some sense explanations were being produced from an understanding of the program, then actually writing the program in the first place would not be much more difficult. I suspect this is true in general. It seems that the primary difficulty in both explanation and automatic programming is a knowledge representation problem, and that the kinds of knowledge to be represented in both cases are similar so that a solution to one case makes the other much easier. Furthermore, if this conjecture is correct, it implies that we are not likely to find easier approaches to explanation than the one presented here (if we require that our explanations be based on an understanding of the program as opposed to, say, canned-text.)

7.4 Is a Top-down Approach Really Necessary?

The XPLAIN system can produce good justifications in part because it has access to the refinement structure produced by the automatic programmer in a top-down fashion. A natural question to ask is whether a bottom-up approach might not work equally well. In other words, one could envision a system that analyzed an existing program structure into higher principles, and explained it at that level. This system would need to employ knowledge structures much like the domain principles and domain model, but they would be used in reverse to parse the existing performance program into a parse tree (which would correspond to XPLAIN's refinement structure). [12] This approach is enticing: it seems that if it can be made to work in general then any program can be explained whether or not it was written with explanation in mind. While such an approach might be attractive in principle, I feel there are several obstacles that make its implementation difficult. First, as was pointed out earlier, the process of writing a program is a process that distills "how-to-do" something out of a much larger body of knowledge. Given that, the analyzer will not be able to explain a program without knowledge structures similar to the domain principles and domain model used by XPLAIN, and furthermore, these structures will have to be similar in both size and scope to those used by XPLAIN. While XPLAIN works deductively this recognizer would have to work by induction and the possibility of ambiguity would exist. In the XPLAIN system, a major effort involved figuring out how the domain principles and domain model should be represented. Once they existed, it was relatively easy to get the program writer to use them to write the performance program. Since both require similar knowledge structures, and once they exist it is easy to synthesize the performance program, the top-down approach would appear to have the edge.

7.5 Limitations and Extensions of the XPLAIN System

While the explanations presented in this paper provide an indication of the power of the automatic programming approach to explanation, they do not exhaust its possibilities. The current system can be extended in several areas:

---

12. See also [Clancey79] for a discussion of this approach.

7.5.1 What Can the Current Implementation Do?

The current domain model and domain principles contain enough knowledge to generate all the examples within this paper. They can also produce additional examples, although these are quite similar to those that appear here. There are three things that would have to be done to complete the implementation of the Digitalis Advisor. First, it would be necessary to implement routines to assess therapeutic improvement. These should not be too difficult because they can be very similar to the routines that assess toxicity. Second, it would be necessary to recapture from the old Digitalis Advisor domain principles to adjust the dose based on the therapeutic and toxic assessments. Third, it would be necessary to implement various utility functions for gathering data and the pharmacokinetic model of digitalis excretion. While there would be a fair amount of programming involved, I do not foresee any major conceptual hurdles. Once this implementation was completed, the domain principles of that program could be used with different domain models to develop similar consulting programs (i.e. programs that offered advice about therapy with various drugs).

7.5.2 Improved Answer Generators

Additional answer generators could be employed to provide the user with: 1) improved access to the domain model so that the domain model itself could be explained as well as its use in the development of the program; 2) improved explication of the decisions made by the automatic programmer; 3) an ability to assess the significance of the program's recommendations.

1) Currently, the explanation routines make use of the domain model to justify a piece of program structure. It would be nice (particularly in a teaching environment) to have answer generators which focused on the domain model so that a user could enhance his understanding of the domain. In addition, it would not be particularly difficult to cross-reference the domain model with the refinement structure to indicate where the domain knowledge was used in the program. This would allow the system to answer questions such as, "How does the system take increased calcium into account?" The answer would be produced by finding those places in the system where the concept increased calcium was used[13] and then displaying the appropriate pieces of code (see also [Swartout77a]).

2) The current system has no ability to explain domain or refinement constraints. In part,

13. For example, increased calcium could be a match for a pattern variable used in a domain rationale or as an argument to a domain constraint.

this is because the implementation of the XPLAIN system has concentrated on offering explanations to medical users and it was felt that the constraints have more of a computer than medical viewpoint. But that is not entirely correct. Recall that when the system was refining the split-join associated with anticipating toxicity it was necessary to assure that all the factors involved were at least causally additive. Whether or not the factors are additive is a question that clearly involves medical knowledge, and it is something which should be explainable to a medical audience in terms of its medical significance.

3) The system should also be able to explain the advice of the performance program in terms of its medical significance. For example, the advisor might conclude that no digitalis should be given for 3 hours and then 0.25 mg should be administered. If the advice was given at 11pm, the patient would have to be awakened at two in the morning if the attending physician wished to follow the program's recommendation to the letter. However, since digitalis has a relatively long half-life, the precise timing of doses (within a few hours) is not thought to be terribly important. In this case, the inconvenience and discomfort involved in waking the patient would probably dictate that the patient receive the drug at an earlier time. While we could program the system so that it does not give drugs during sleeping hours, it seems that that approach might eventually result in a program which knew substantially more about hospital procedure than about digitalis therapy. A better approach might be one where the system could indicate to the user the importance of its recommendations. For example, in this case, the system could mention that a variation of a few hours in drug administration would not be significant.

7.5.3 Telling White Lies

Currently, XPLAIN can describe what a performance program it creates does and why it does it at various levels of abstraction by describing the methods it uses, the refinement structure, the domain principles and the domain model. While it can leave out details based on viewpoint or by using a higher abstraction, it does not deliberately distort its explanations. Yet sometimes human teachers do exactly that to make their explanations easier to understand. That is, to introduce a new concept, a teacher may deliberately over-simplify things and give an explanation which is in fact wrong but easier to understand and close enough to the truth that the student can easily understand the correct view later once he understands the approximation the teacher presents. These "white lies" represent a kind of explanation that cannot be delivered by systems that just present (at some level of detail) the reasoning paths followed by a problem solver (or automatic programmer).

But where do white lies come from? Sometimes a teacher may create them from scratch, but often they are just earlier versions of what was thought at the time to be the complete, final version of the theory, program or whatever. For example, the old Digitalis Advisor used to adjust the dose for sensitivities using a simple threshold model: if the level of serum potassium (say) was below a certain threshold. the dose was reduced by a fixed percentage. The most recent version of the Digitalis Advisor makes a sliding reduction depending on how depressed the serum potassium is. Since the threshold model is more understandable, when describing the Digitalis Advisor, one might present it first to give the user a general idea of what happens before describing the sliding reduction that is actually used.

It seems that it could be very helpful in providing better explanations if the XPLAIN system adopted a similar technique. That is, suppose that someone discovered that a domain principle had to be modified to give better results. Currently, he would make the modification and re-run the program writer to get a new performance program which incorporated the change. The old version of the domain principle and performance program would be discarded. What we are suggesting here is that rather than throwing away old versions of the performance program, it might be useful if the program writer kept them around and noted the differences between the refinement of the old program and the new and where these differences arose (i.e. new principle, different domain model, etc.). The explanation routines could then use the old program fragments as a source for white lies and after the old version was understood, the difference links could be used to indicate how things really worked. Additionally, recording the changes between versions would allow the system to offer effective explanations about those changes to a user who had not used the system for a while. To continue the example above, suppose a user who had last used the performance program when it made reductions by a threshold used the new version with sliding reductions. If his patient were only slightly hypokalemic, he might wonder why the reduction for decreased potassium was much smaller than before. The system could justify the difference only if it has access to the differences between the two versions and the reasons for those differences. Of course, the system would have to be careful. Sometimes new program fragments would result from new insights into the problem, leading to a better *and* simpler program. In that case, referring to the old program would gain nothing for expository purposes, although it could still be used for explaining differences between the old and new versions.

7.5.4 Telling the User What He Wants to Know

While the current system has a limited ability to tailor the explanation to the interests of the user and to model what has been explained to him, the quality of the explanations could be substantially improved if the results of other research efforts could be integrated with the XPLAIN system. These include: 1) having the system model what it believes the user knows [Genesereth79], 2) developing tutorial strategies giving the system a more global view of its interaction with the user and allowing it to take part in directing it [Carr77a, Clancey79], 3) on the opposite end of the scale, improving the low level English generators so they are more firmly grounded on linguistic principles [McDonald80, Mann80] and 4) improving the system's understanding of its own explanatory capabilities and the user's question so that it can reformulate the user's request into what it can deliver [Mark80].

8. A Summary of Major Points

First, we have argued that to be acceptable, consultant programs must be able to explain what they do and why. Second, we have described the various ways that traditional approaches fail to provide adequate explanations and justifications. Major failings include: 1) the inability of such approaches to justify what the system is doing because the knowledge required to produce justifications is not represented within the system, and 2) a lack of distinction between steps required just to get the computer-based implementation to work, and those that are motivated by the application domain. Third, we have outlined an approach which captures the knowledge necessary to improve explanations. This involves using an automatic programmer to generate the performance program. As the program is generated, a refinement structure is created which gives the explanation routines access to decisions made during the creation of the program. The improvement in explanatory capabilities that is achieved is due more to the availability of this refinement structure than to the use of more sophisticated English generation functions.

9. Acknowledgments

I would like to thank the members of the MIT Clinical Decision Making Group, particularly Ramesh Patil, Ken Church, Bill Long, Harold Goldberger, and especially my advisor, Peter Szolovits, for insightful comments and suggestions that greatly aided this research. I would also like to thank Bob Balzer, Lee Erman, Mike Fehling, Jack Mostow and Norm Sondheimer all of ISI, for their suggestions which facilitated the writing of this paper.

10. References

[Barstow77] Barstow, D., "A Knowledge-Based System for Automatic Program Construction," Proceedings of the Fifth International Conference on Artificial Intelligence, 1977

[Balzer77] Balzer, R., Goldman, N., Wile, D., "Informality in Program Specifications," Proceedings of the Fifth International Conference on Artificial Intelligence, 1977

[Brown75] Brown, J.S., Burton, R.R., Zydbel,F., "Multiple representations of knowledge for tutorial reasoning," in **Representation and Understanding** D.G. Bobrow and A. Collins (eds) Academic Press, New York, 1975

[Carr77a] Carr, B. and Goldstein, I.P., "Overlays: a Theory of Modelling for Computer Aided Instruction," MIT AI Laboratory Memo 406, February 1977.

[Carr77b] Carr, B., "Wusor II: A Computer Aided Instruction Program with Student Modelling Capabilities," MIT Artificial Intelligence Laboratory Memo 417, May 1977

[Clancey79] Clancey, W.J., "Transfer of Rule-based Expertise Through a Tutorial Dialogue", Stanford University, Department of Computer Science, STAN-CS-79-769. 1979

[Clancey81] Clancey, W.J., "Neomycin: Reconfiguring a Rule-based Expert System for Application to Teaching," Proceedings of the Seventh International Conference on Artificial Intelligence, 1981

[Davis76] Davis. R., "Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases," PhD thesis, Stanford Artificial Intelligence Laboratory Memo 283(1976).

[Genesereth79] Genesereth, M.R., "The Role of Plans in Automated Consultation," *Proceedings of the Sixth International Conference on Artificial Intelligence*, 1979

[Gorry78] Gorry, G. A., Silverman, H., and Pauker, S. G., Capturing Clinical Expertise: A Computer Program that Considers Clinical Responses to Digitalis, *American Journal of Medicine* **64**:452-460, (March 1978).

[Green79] Green, C.C., Gabriel, R.P., Kant, E., Kedzierski, B.I., McCune, B.P., Phillips, J.V., Tappel, S.T., Westfold, S.J., "Results in Knowledge Based Program Synthesis," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 1979

[Hawkinson80] Hawkinson, L.B., "XLMS: A Linguistic Memory System," MIT Laboratory for Computer Science TM-173, 1980

[Jelliffe70] Jelliffe R.W., Buell J., Kalaba R. et al, "A Computer Program for Digitalis Dosage Regimens," Math. Biosci. 9:179-193, 1970

[Jelliffe72] Jelliffe R.W., Buell J, Kalaba R, "Reduction of digitalis toxicity by computer-assisted glycoside dosage regimens," Ann. Intern. Med. 77:891-906, 1972

[Kastner82] Kastner, J. K., Weiss, S.M., Kulikowski, C.A, "Treatment Selection and Explanation in Expert Medical Consultation Application to a Model of Ocular Herpes Simplex," Proceedings of MEDCOMP82, p.420-427, 1982

[Long77] Long, W.J., "A Program Writer," MIT Laboratory for Computer Science, TR-187, 1977

[Mann80] Mann, W.C., Moore, J.A., "Computer as Author—Results and Prospects," USC Information Sciences Institute ISI/RR-79-82, 1980

[Manna77] Manna, Z., Waldinger, R., "The Automatic Synthesis of Systems of Recursive Programs," Proceedings of the Fifth International Conference on Artificial Intelligence, 1977

[Mark80] Mark, W., "Rule-Based Inference in Large Knowledge Bases," *Proceedings of the First Annual National Conference on Artificial Intelligence*, 1980

[Martin79] Martin, W.A., "Roles, Co-Descriptors and the Formal Representation of Quantified English Expressions," MIT Laboratory for Computer Science TM-139, September 1979

[McDonald80] McDonald, D.D., "Natural Language Production as a Process of Decision-making Under Constraints," MIT PhD Thesis, 1980

[Minsky75] Minsky, M., "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P.H. Winston (ed), McGraw-Hill, 1975

[Peck73] Peck C.C., Sheiner L.B. et al: "Computer-assisted Digoxin Therapy," *New England Journal of Medicine* 289:441-446, 1973.

[Pauker76] Pauker, S.G., Gorry, G.A., Kassirer, J.P., and Schwartz, W.B., "Toward the Simulation of Clinical Cognition: Taking a Present Illness by Computer," The American Journal of Medicine 60:981-995 (June 1976).

[Pople77] Pople, H.E. Jr., "The Formation of Composite Hypotheses in Diagnostic Problem Solving: an Exercise in Synthetic Reasoning," Proceedings of the Fifth International Joint Conference on Artificial Intelligence (1977).

[Sheiner72] Sheiner L.B., Rosenberg B., Melmon K., "Modelling of Individual Pharmacokinetics for Computer-aided Drug Dosage," *Computers and Biomedical Research* 5:441-459, 1972

[Shortliffe76] Shortliffe, E.H., **Computer Based Medical Consultations: MYCIN,** Elsevier North Holland Inc. (1976)

[Silverman75] Silverman, H., "A Digitalis Therapy Advisor," MIT Project MAC TR-143, 1975

[Swartout77a] Swartout, W.R., "A Digitalis Therapy Advisor with Explanations," MIT Laboratory for Computer Science TR-176, February 1977

[Swartout77b] Swartout, W.R., "A Digitalis Therapy Advisor with Explanations," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, August 1977

[Swartout81] Swartout, W.R., "Producing Explanations and Justifications of Expert Consulting Systems," MIT Laboratory for Computer Science Technical Report TR-251, January 1981.

[Swartout82] Swartout, W.R., Balzer, R., "On the Inevitable Intertwining of Specification And Implementation" *Communications of the ACM* 25:7 438-439 (1982)

[Weiss78] Weiss, S.M., Kulikowski, C.A., Amarel, S., Safir, A., "A Model-Based Method for Computer-Aided Medical Decision-Making," *Artificial Intelligence*, 11:1,2 (1978)

[Winograd71] Winograd, T., "A Computer Program for Understanding Natural Language," MIT Artificial Intelligence Laboratory TR-17, 1971

END

FILMED

8-83

DTIC