1

AMERICAN MICROFILMS
MicroD
SERIALS ACQUISITION
300 NORTH ZEEB RD.
ANN ARBOR MI 48106

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

RATIONALE

A Tool for Developing Knowledge-Based Systems that Explain by Reasoning Explicitly

by

Suhayya Abu-Hakima, B.ENG.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University

April, 1988

© copyright

1988, Suhayya Abu-Hakima

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-46286-8

The undersigned hereby recommends to the Faculty of Graduate Studies and Research
acceptance of the thesis,

" RATIONALE: A Tool for Developing Knowledge-Based Systems
that Explain by Reasoning Explicitly "

submitted by SUHAYYA ABU-HAKIMA, B.ENG., in partial fulfillment of the
requirements for the

degree of Master of Engineering

F. Oppacher
Thesis Supervisor
Department of Computer Science

S.A. Mahmood
Chairman
Department of Systems and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

April, 1988

Abstract

Explanations generated by current expert systems are often terse, include bookkeeping information irrelevant to most users and are not well integrated with the reasoning processes of the expert system.

The thesis presents a tool, RATIONALE, for building expert systems that explain by reasoning explicitly. RATIONALE is an expert systems development tool - implemented in Prolog - that provides a developer with an integrated environment for tying together the processes of knowledge acquisition, expert system construction and explanation of system reasoning.

RATIONALE integrates explanations with reasoning by using a causal and an explicit knowledge representation of the domain. RATIONALE's inference mechanism reasons about this explicitly represented knowledge using an explicit reasoning algorithm. Reasoning proceeds by constructing an object-oriented classification tree whose root object contains the most general diagnosis of the expert system. Guided by a focusing algorithm, the classification tree branches into more specific hypotheses that explain the more detailed symptoms provided by the user. As the expert system is used, the classification tree also forms the basis for a dynamically generated explanation tree.

Acknowledgements

I would like to thank Franz Oppacher for guiding me throughout this work. I gratefully acknowledge the Knowledge Technology group at BNR, specifically Dick Peacocke for giving me the opportunity to pursue my Masters while employed there, and Innes Ferguson who was always there to bounce ideas off. I am also very grateful for the BNR Scholarship which was awarded to me as further incentive to pursue this work. A special thanks is also extended to my friend Arthur van Hoff who was a great help in editing the many final versions. And foremost, I am very thankful to my much loved husband Ken Grigg who cheered me on throughout.

Table Of Contents

1. Introduction.....	1
2. Current Approaches to Explanation.....	5
2.1 Successful Rule Tracking and Canned Text Explanations.....	5
2.2 Executed Program Trace and Template Explanations.....	6
2.2.1 MYCIN.....	6
2.2.2 GUIDON and NEOMYCIN	14
2.3 Reasoning Trace and Deep Explanations.....	21
2.3.1 BLAH.....	22
2.3.2 XPLAIN and EES.....	26
2.3.3 MDX, PATREC and CSRL.....	34
2.4 Adequacy Criteria for Knowledge-Based Explanations.....	48
3. RATIONALE.....	55
3.1 Design.....	59
3.2 State Generator.....	62
3.3 Frame Generator	64
3.4 Window Generator	68

3.5	Explanation Generator.....	76
3.5.1	RATIONALE's Reasoning Algorithm.....	76
3.5.2	Explaining the Reasoning Strategies.....	79
3.5.2.1	Session Sensitive Questions.....	79
3.5.2.2	System Capability Questions.....	81
3.5.2.3	Generated Explanations.....	83
4.	Implementation of RATIONALE	89
4.1	Example Domain.....	91
4.2	Frame Generator	93
4.2.1	Top Goal Frame	93
4.2.2	Subgoal Frame.....	95
4.3	Explanations.....	100
4.3.1	State Traversal	100
4.4	Explaining the Selected Problem.....	110
4.5	Explaining the Reasoning Strategies.....	123
4.5.1	Display Questions	124
4.5.1.1	Event Questions.....	125
4.5.1.2	Hypothetical Questions.....	126
4.5.1.3	Ability and Factual Questions.....	126

4.5.2 Generate Explanations	128
4.5.2.1 Event Explanations.....	128
4.5.2.2 Hypothetical Explanations	148
4.5.2.3 Ability Explanations	150
4.5.2.4 Factual Explanations.....	156
5. Summary	162
5.1 Concluding Remarks	162
5.2 Discussion and Extensions to RATIONALE	163
5.2.1 Modeling Domain Knowledge	163
5.2.2 Learning by Generating Intermediate Hypotheses.....	164
5.2.3 User Modeling in Explanation	164
5.2.4 Explaining Interactions of Multiple Domains.....	165
5.2.5 Explaining Evidential Reasoning	165
References.....	167
Appendices.....	174
Appendix A: Object Definition of ACME Workstation	174
Appendix B: RATIONALE Explanation Examples	184
B1: Main Menu Options.....	184
B2: Event Question Explanations.....	191
B3: Hypothetical Question Explanations	218
B4: Ability Question Explanations	228
B5: Factual Question Explanations.....	231

List of Figures

Figure 1-1	Depth in Knowledge-Based Systems.....	2
Figure 2-1	MYCIN - A Production-Based Consulting System with Explanation Capability.....	7
Figure 2-2	Relating MYCIN'S Rule Ordered Dialog to Organism Hypotheses.....	12
Figure 2-3	RATIONALE's Integrated Explanation Capability.....	13
Figure 2-4	Organization of BLAH.....	25
Figure 2-5	XPLAIN's knowledge Separation	26
Figure 2-6	General Structure of XPLAIN.....	28
Figure 2-7	Planned Structure of EES	32
Figure 2-8	Conceptual Structure of Cholestasis	36
Figure 2-9	Remainder of the Conceptual Structure of Cholestasis	37
Figure 2-10	Collecting Evidence in MDX.....	41
Figure 2-11	Comparison of Explanation Facility Adequacy Criteria.....	50
Figure 3-1	Hypothesis Tree for ACME Workstation Example.....	56
Figure 3-2	RATIONALE's Top Level Structure.....	61
Figure 3-3	RATIONALE's Finite State Machine	63
Figure 3-4	RATIONALE's Generalized Top Goal	65
Figure 3-5	RATIONALE's Generalized Subgoal.....	67
Figure 3-6	Defining a New Expert System in RATIONALE.....	69

Figure 3-7	Defining a New Top Level Goal in RATIONALE	70
Figure 3-8	Defining a Subgoal in RATIONALE.....	71
Figure 3-9	Selecting an Observed Problem in an expert system run in RATIONALE	73
Figure 3-10	Selecting Relevant Symptoms of Observed Problem	74
Figure 3-11	Example Advice for Observed Problem in RATIONALE	75
Figure 3-12	RATIONALE's Reasoning Algorithm.....	77
Figure 4-1	ACME Workstation Hypothesis Hierarchy	92

Glossary of Terms

causal model

a domain model which represents the causal relations among various actions and events.

synonyms: functional model

classification

a method of finding a solution for a particular problem in an expert system by matching observed symptoms to those required to satisfy a problem in a hierarchy of problems.

explanation

information justifying a specific course of action or describing the static knowledge in an expert system.

explanation trace

the reasoning chain which holds the dynamic knowledge used to conclude a particular result. The explanation trace can be: a list of successfully triggered rules in a rule-based system; or a dynamic goal tree with indications of what goals succeeded or failed in a hypothesis-based system.

synonyms: mechanism tracking reasoning, history tree, reasoning trace, reasoning tree

executed program trace

a trace similar to a debugging trace of a conventional program. The executed program trace is often used in a rule-based system as an explanation trace. It often contains information that is not directly relevant to explanation including bookkeeping information such as which rule number was triggered.

refined trace

an executed program trace which holds only the reasoning information relevant for explanation.

synonyms: explicit trace, explicit reasoning trace

canned text explanations

refers to the text generated in response to a user question. Canned text explanations are strings of text that are attached to rules and output as explanations to user questions when the rule is triggered successfully.

template-based explanations

refers to the text generated in response to a user question. This form of explanation connects pieces of canned text to dynamic variables that are instantiated from the user's current context.

deep explanations

explanations generated on the basis of a causal or functional model of the domain.

shallow explanations

explanations that are not based on a functional model of the domain, rather they are based on loosely related knowledge of the domain.

event explanations

a category of explanations that uses dynamic knowledge of the user's current session to answer questions about the reasoning strategies of the system, typically known as *why* and *how* questions. These questions include: why symptoms are (are not) used; why results are (are not) concluded; and how results are concluded.

synonyms: session sensitive explanations

hypothetical explanations

a category of explanations that uses both static and dynamic knowledge in the user's current session to answer *what-if* questions. These questions include: what-if new symptoms are added to the current context; and what-if new symptoms and a new context replace the previous symptoms and context.

ability explanations

a category of explanations that uses static knowledge to answer user questions about the reasoning strategies of the system. These questions include how are contexts established with observed symptoms.

factual explanations

a category of explanations that uses static knowledge to answer user questions about facts in the system. These questions include what are the symptom values that satisfy the conditions on the reasoning strategies.

Diagnosis

a method of finding a solution in an expert system, closely related to classification, in which observed symptoms are matched against symptoms that account for equipment faults (or diseases in medical applications).

domain model

a model of a problem domain that incorporates facts, assumptions, beliefs and heuristics about the various aspects of the problems that make up the domain.

domain principles

prescriptive knowledge of a domain.

frame

a knowledge representation scheme that associates an object with a collection of attributes (such as facts or rules). Each attribute is stored in a frame slot. A frame is a set of slots related to a specific object (it is similar to a property list, schema or record used in conventional programming).

top goal frame

a knowledge representation scheme used to represent the conditions of the top goal in a goal hierarchy.

subgoal frame

a knowledge representation scheme used to represent the conditions of any goal below the top goal in a goal hierarchy. The attributes of the frame include the conditions that make a goal active as well as its position in the goal hierarchy relative to its parent and sibling goals.

goal

the end to which problem solving aspires. In an expert system goal hierarchy, goals are organized into a tree whose leaves represent problems that a user observes and requires advice on.

synonyms: hypothesis, problem, object

subgoal

a goal that falls under a parent goal. The heuristics of a subgoal are dependent on those of its parent.

synonyms: subproblem

top goal

the root or top most goal in a goal hierarchy.

synonyms: top level goal, top level hypothesis

top level subgoal

a goal immediately below the root or top most goal in a goal hierarchy. The top level subgoal has associated heuristics which are inherited by the goals immediately below it.

synonyms: parent subgoal

refinement subgoal

a goal one or more levels below the top level subgoal in a goal hierarchy. The refinement subgoal has associated heuristics which are inherited by the goals immediately below it.

synonyms: sibling subgoal

active goal

a goal whose conditions are partially satisfied by observed user symptoms. Once a goal becomes active, all its conditions must be satisfied before the goal hierarchy can be descended and any of the goal's sibling goals can be activated. The sibling goals are activated to search for the goal which provides the best match with the user's observed symptoms.

synonyms: active hypothesis, enabled hypothesis

goal hierarchy

a top-down tree of goals for which heuristics can be applied at every level. Each goal may represent a problem which has associated conditions that represent a class of problems. Thus, to traverse to a lower level of the goal hierarchy it is necessary to satisfy the conditions of the parent goal.

synonyms: hypothesis hierarchy, hypothesis tree, domain hierarchy, problem hierarchy, conceptual hierarchy, refinement structure, taxonomic classification hierarchy

inference

the process by which new knowledge is derived from other known knowledge.

synonyms: reasoning

inductive inference

a mode of reasoning which starts with given facts and concludes with general hypotheses or theories from which initial facts are re-derived via deductive inference.

deductive inference

in formal logic, the derivation of a logical consequence from a specific set of premises. A truth-preserving transformation of assertions.

abductive inference

is the process of reasoning about data (specific to a particular situation) to formulate an explanatory hypothesis that best accounts for the data. Abductive inference is accomplished by assembling explanatory subparts into a unified explanatory hypothesis.

implicit reasoning

is the process of reasoning in an expert system *without* using strategic information about the domain knowledge. This includes the strategy behind encoding the domain knowledge. For example, in a rule-based system, this includes the knowledge used to write rules, including their justification, ordering, and planned execution.

synonyms: shallow reasoning

explicit reasoning

is the process of reasoning in an expert system using strategic information about the domain. This form of reasoning is often based on a functional model of the domain represented as a goal or hypothesis hierarchy. The system reasons using explicit strategies that are directly related to the functional model of the domain.

synonyms: deep reasoning

hierarchical reasoning

is the process of reasoning using a distributed control method in which the controlling processes are arranged in a hierarchy and where the results of reasoning about the higher level elements in the hierarchy controls the reasoning at the lower level elements.

synonyms: classificatory reasoning

refinement

is the process of traversing a goal hierarchy by satisfying the conditions of a higher level goal to activate and satisfy the conditions of a lower level goal. The refinement process stops only when the conditions of the lower level goal cannot be satisfied or when the bottom goals (those at the lowest level) of the hierarchy are reached.

synonyms: focusing, refinement algorithm

knowledge

an integrated collection of facts and relationships. The quantity and quality of knowledge processed by an individual or a program can be judged by the variety of situations in which the individual or program obtains successful results.

domain knowledge

knowledge about a particular topic or problem area used to advise an expert system user (e.g. knowledge of infectious disease or knowledge of computer equipment faults).

static knowledge

facts, assumptions, beliefs and heuristics about a domain that are encoded as the domain knowledge base.

dynamic knowledge

assertions, conclusions, and goals (active or satisfied) which are relevant to a particular domain, and which account for symptoms observed by a user requiring advice from an expert system.

descriptive knowledge

facts and textbook-type knowledge which describes a domain (see description of XPLAIN in section 2.3.2).

prescriptive knowledge

- ◊ assumptions, beliefs and 'how-to' heuristics which describe the methods a human expert uses to solve a problem in a particular domain. (see description of XPLAIN in section 2.3.2).

synonyms: procedural knowledge

reasoning strategies

strategies that are used by the inference mechanism to reason about the domain.

enabling conditions

conditions (problem enabling symptoms and a required percentage of match) that partially satisfy a goal in the goal hierarchy. The enabling conditions activate a branch of the goal hierarchy for subsequent refinement. These conditions can be viewed as providing positive evidence for belief in a particular hypothesis.

synonyms: triggering conditions

invalidating conditions

conditions (problem invalidating symptoms and a required percentage of match) that override the enabling conditions of a goal. The invalidating conditions de-activate an activated branch of the goal hierarchy. These conditions can be viewed as providing negative evidence for belief in a particular hypothesis.

tolerance conditions

conditions (problem tolerance symptoms and a required percentage of match) that override the invalidating conditions of a goal. These conditions can be viewed as special case conditions which when satisfied provide exceptions for negative evidence. The tolerance conditions re-activate a de-activated branch of the goal hierarchy, provided that the de-activation was caused by the invalidating conditions.

synonyms: special case conditions

refinement conditions

conditions (problem refinement symptoms and a required percentage of match) placed on a higher level goal to activate a lower level goal. A goal may have several subgoals that it can be refined to. Each of these subgoals has an associated set of sufficient symptoms for refinement and a required percentage of match.

synonyms: sufficient conditions for refinement

alternate problem conditions

conditions (alternate problem symptoms and a required percentage of match) placed on a goal to activate a goal in another branch of the goal hierarchy. A goal may have several alternate goals that it can activate. Each of these goals has an associated set of sufficient

symptoms for activating the alternate problem and a required percentage of match. The alternate problem branch is refined in a manner identical to that of the original problem.
synonyms: alternate problem conditions

synthesis

a method of finding a solution which is constructed from several subparts in an expert system .

Chapter 1

Introduction

The ability to construct explanations is widely regarded as a crucial feature of knowledge-based systems, and much has been expected of it. Explanations have been variously described as: a means of debugging knowledge-based programs and of providing dynamic justifications for unexpected system behaviour; a means of verifying the reasoning of the domain expert; and as training aids for novice users [Hasling84]. Hayes has defined them in [Hayes83] as "*a means of capturing the knowledge-based program behavior of a system that reasons symbolically*".

It is obvious that such strong expectations are very difficult to live up to and that existing, off the shelf explanation facilities fall far short of realizing them. It is less obvious why this should be so, but it is likely that the reason lies in the fact that an explanation facility is usually added to an inference engine as a mere 'afterthought' instead of being built first or at least concurrently with it. By contrast, RATIONALE is designed to reason explicitly for the purpose of clearly explaining its reasoning strategies, i.e. its inference engine coincides with its explanation module. By thus aligning the functioning of the inference engine with that of the explanation module, the latter has immediate access to all the domain knowledge and strategic information that drives the former.

Consider the approach taken in building knowledge-based systems that can reason with some depth. In Figure 1-1, compiled heuristic rules are placed in the static knowledge

base. Once a user presents an observed problem to the system, a match between the rules that describe a range of problems, and the facts that describe the observed problem is attempted. This is reasoning by forward chaining through the knowledge base of rules. If the match succeeds, a rule is triggered by the inference engine to update the current *state of the world*. The current state maintains the dynamic knowledge of the case data for the current session. The reasoning behind triggering the rule is limited to the fact that the match succeeded, without knowledge of the reasoning mechanisms behind it [Bobrow86]. Such shallow reasoning provides for equally shallow explanations.

Figure 1-1: Depth in Knowledge-Based Systems

Had the match failed, a causal or functional model of the domain may then be taken as a fallback and examined for any indirect links to the current case. It is such a domain model which provides the system reasoning with some depth [Chandrasekaran 83]. The search for a link is performed with a hill climbing algorithm which fires rules that may or may not be applicable, thus resulting in a slow unguided search. This is where the

difference between the usual approach taken in deep reasoning systems differs from RATIONALE's. In RATIONALE, the search strategy is guided by a focusing algorithm. The algorithm serves to guide the search according to the heuristics of the domain model that relate the domain symptoms for a problem with those observed symptoms. The decisions for guiding the search strategy are performed at every node in the hypothesis tree. Every node is representative of a hypothesis within the functional model of the domain. For a hypothesis to be refined further, a degree of match between the observed world of the user and that of the hypothesis has to be satisfied. The search necessary for the system to reason is guided by strategies known to the inference mechanism. Reasoning continues only if the inference is explainable according to the functional model. RATIONALE's approach to explaining by reasoning explicitly provides for a powerful explanation capability.

Accordingly, RATIONALE's explanation module is able to do the following:

first, answer basic user questions - at user-determined levels of detail - about strategic or heuristic decisions and, in general, about the knowledge-based behaviour of the system, and

second, give explicitly reasoned explanations - again at varying levels of detail - to user questions on the basis of the system's domain model.

Moreover, make it easy for the user to request explanations, RATIONALE provides a 'map' of the available explanations by means of menus.

RATIONALE is both a shell for building diagnostic expert systems as well as an object-oriented tool that integrates into one the mechanisms of knowledge acquisition, expert system construction and explanation of system reasoning. RATIONALE is also extendable to problems other than diagnosis, one such example is the problem of synthesis,

another expert system problem which can be reasoned about explicitly and one whose reasoning is based on classification.

In the remainder of the thesis: Chapter 2 surveys recent research into explanation facilities and compares some traditional explanation facilities with those used in RATIONALE. It also outlines the key design considerations in constructing expert systems that can explain their reasoning by integrating explanations with their reasoning processes. RATIONALE's overall design is described in Chapter 3. RATIONALE is implemented in Quintus Prolog, initially running on a Xerox 1109 workstation and has since been ported to a Sun 3/60. Its implementation is detailed in Chapter 4 (example sessions are included in Appendix B). Future extensions as well as discussions of reasoning-based explanation conclude the thesis in Chapter 5.

Chapter 2

Current Approaches to Explanation

Explanation facilities have generally been added to knowledge-based systems to explain the inferences that occur as the system reasons. Explaining in such systems takes the form of tracking inferences. The mechanism which tracks reasoning in order to explain is the main distinguishing characteristic of current facilities. The next three sections describe the explanation types that are usually associated with three widely used tracking mechanisms.

2.1 Successful Rule Tracking and Canned Text Explanations

Canned text explanations are used in most systems whose tracking mechanisms only maintain a list of successfully triggered rules or satisfied goals. Canned text explanations are by far the most common type of explanation in current expert systems because they are so easy to implement.

Strings of canned text are attached to all rules so that premises and conclusions are explainable in terms of the programmer's predicted execution of the program, and the text string associated with any successful rule is emitted when an explanation is requested. An expert system builder is faced with the difficult tasks of having to try to anticipate the many possible questions a user might want to ask and of maintaining the consistency of the evolving system with the canned text. Such an approach is bound to ignore the system's reasoning about its recommendations and strategic choices. Moreover, it does not provide explanations for decisions taken in refining hypotheses according to the validity or invalidity of the user's observed symptoms versus the system's expected symptoms.

2.2 Executed Program Trace and Template Explanations

2.2.1 MYCIN

The executed program trace is the basis for another common type of explanation in expert systems. This approach to explanation was pioneered by the builders of MYCIN [Scott 77]. As illustrated in Figure 2-1, MYCIN has static knowledge and dynamic knowledge. The static knowledge consists of production rules and general facts about its consultation domain, that of advising physicians on the method of treatment for patients with microbial infections. The dynamic knowledge is the knowledge gathered in a production run, an infectious disease therapy consultation in which MYCIN advises the doctor about the treatment of a patient. This knowledge encompasses dynamic facts about the problem input by the user as well as current assertions or deductions made by the

system. The general and dynamic facts as well as the current assertions make up the knowledge or *data base* accessed by both, the rule interpreter as well as the *explanation capability* mechanism. Inferences made by the rule interpreter are output to the user as consultation advice and are used to prompt the user for additional facts about the problem at hand.

Figure 2-1: MYCIN - A Production-Based Consulting System with Explanation Capability

The explanation mechanism is used to answer the user's questions about deductions in the current session. It uses both the static as well as dynamic knowledge for explanation. Note however, that it does not use knowledge of the rule interpreter or inference mechanism for explanation. The explanation mechanism can only access the static and

dynamic knowledge components of MYCIN, thus excluding explanation of the reasoning mechanism.

MYCIN tracks successfully triggered rules in forward chaining and it maintains a goal-oriented trace in backward chaining. Knowledge is represented in attribute-object-value triples. In response to the system's questions, the users input details about objects with which MYCIN establishes *contexts* (ie. the patients, their infections, cultures taken and drugs administered). Contexts are uniquely named and their interrelations are represented in the *context tree*. The consultation dialog is used to establish values of the attributes or clinical parameters of contexts (eg. the *age* of the patient, the *identity* of the organism with *streptococcus* as the possible value, etc.). These clinical parameters are established by asking the user or by inference from the rules.

MYCIN can explain its procedural knowledge in two ways. The first, matches user questions to conceptual primitives attached to static rules. The primitives have associated translation templates used to output explanations. The second form of procedural explanation uses specialists or demons dedicated to answering a unique question. For example:

RULE 009

**PREMISE: (\$AND (SAME CNTXT GRAM GRAMNEG)
(SAME CNTXT MORPH COCCUS))**

ACTION: (CONCLUDE CNTXT IDENTITY NEISSERIA TALLY 800)

[Translation:

**If : 1) The gram stain of the organism is gramneg, and
2) The morphology of the organism is coccus**

**Then: There is strongly suggestive (0.8 certainty) that the identity of the organism is
Neisseria]**

In the example, the demon was asked by the user to find out how the identity of a *NEISSERIA* organism is found. The demon would find the rules that conclude *IDENTITY* and select the ones applicable to the current context.

The explanation specialists access a record of the consultation by using MYCIN's *history tree*. The tree is goal-directed and it forms a dynamic trace of the current consultation. Each node in the tree represents a goal and contains information about rules the system tried to meet the goal. Associated with each goal is a record of whether the rule succeeded or not and what inference it failed on.

MYCIN accepts two forms of user questions for explanation of session reasoning. The first type are described as reasoning-status checker questions [Scott77]. These are *why* and *how* single word questions. *Why* questions are available for the user to ask the system about the relevance of information the system is directed to ask the user for. *How* questions are designed to explain how particular goals are achieved. An example of *why* and *how* explanations proceed as follows:

preceded by the first 14 questions in the consultation

14) Is the patient's illness with ORGANISM-1 a hospital-acquired infection?

**** WHY**

ie. why is it important to determine whether or not the infection with ORGANISM-1 was acquired while the patient was hospitalized?

[1.0] This will aid in determining the category of ORGANISM-1.

It has already been established that

- [1.1] the gram stain of ORGANISM-1 is gramneg, and
- [1.2] the morphology of ORGANISM-1 is rod, and
- [1.3] the aerobicity of ORGANISM-1 is facultative

Therefore, if

- [1.4] the infection with ORGANISM-1 was not acquired while the patient was hospitalized

then

there is strongly suggestive evidence (0.8) that the category of ORGANISM-1 is enterobacteriaceae

ALSO: there is weakly suggestive evidence (.1) that the identity of ORGANISM-1 is *pseudomonas-aeruginosa*
[RULE037]

**** HOW 1.3**

ie. how was it established that the aerobicity of ORGANISM-1 is facultative?

The following were used:

[3.1] RULE027 indicated there is strongly suggestive evidence (0.8) that the aerobicity of ORGANISM-1 is facultative

[3.2] RULE027 indicated there is weakly suggestive evidence (0.2) that the aerobicity of ORGANISM-1 is anaerobic

Since this gave a cumulative CF of (0.8) for facultative, and (0.2) for anaerobic, it has been established that the aerobicity of ORGANISM-1 is facultative

**** HOW 1.1**

ie. how was it established that the gram stain of ORGANISM-1 is gramneg?

You said so [question 11].

An apparent problem with MYCIN's above explanations is the assumed level of intimacy with the rules that the user must have to follow the dialog. Such intimacy cannot always be assumed, especially with less experienced users.

The second form of user questions are processed by MYCIN's *general question-answerer* or GQA facility. The GQA answers template-based questions about both static and dynamic knowledge. A list of question fragments of root words serves as a parser dictionary for user questions. A question is parsed to determine whether it requires a static or dynamic explanation demon. Once the question is identified and the specialist has retrieved the knowledge for the explanation, a template of the answer is filled. The questions accepted by the system are matched against a set of explanation schemas which are filled with appropriate parts of the translated trace of the session.

A template has the following structure:

Question: How do you know the value of <parameter> of <context>?

Answer 1 Template: I used <rule number> to conclude that <parameter> of <context> is <value>. This gave a certainty factor <CF>. The last question asked before the conclusion was <Q-number>.

Answer 2 Template: In answer to <Q-number> you said that <parameter> of <context> is <value>.

Note that the GQA, unlike the reasoning-status checker, does not use the history tree. This restricts GQA explanations to those that can be pattern matched rather than using the goals in the history tree as the basis for the explanations. Although MYCIN uses its trace to explain session sensitive reasoning, it does not explain its reasoning strategies. Much of the strategic information is implicit in the ordering of the rule clauses and is not accessible for explanation [Clancey86; Hasling84]. Therefore, information that is used to write the rules, including justification, ordering, and planning is lost or left implicit. This inexplicable information is, in essence, a large part of the strategy employed to do the diagnosis. The following example illustrates that at a certain level, MYCIN's reasoning is arbitrary [Clancey86].

32] Does J. Smith have a history of NEUROSURGERY?

** NO

33] Does J. Smith live in a crowded environment?

** NO

34] Do you suspect recent alcoholic history in J. Smith?

** NO

35] Is the meningitis a hospital-acquired infection?

** YES

36] Is J. Smith's clinical history consistent with EPIGLOTTITIS?

** NO

37] Is J. Smith's clinical history consistent with OTITIS-MEDIA?

** NO

38] Has J. Smith ever undergone splenectomy?

** NO

39] Is J. Smith a burn patient?

** YES

As it considers types of *organisms*, it considers them on the basis of the order of the rules in the knowledge base rather than hypotheses as illustrated below in Figure 2-2.

Figure 2-2: Relating MYCIN'S Rule Ordered Dialog to Organism Hypotheses

Figure 2-2 illustrates the outcome of ordering MYCIN's questions numbered 32 to 39 according to the hypotheses that motivate them [Clancey86]. For example question 33 about living in a crowded environment is asked in order to apply rule 533, which concludes that the organism is *Neisseria*. The questions pertain to the same goal - determining the organisms therapy should cover - but the rule conclusions relate to different organisms. Neither the sequence of rule applications nor the questions are sorted by organism. Questions 34 and 38 are intended to identify *Diplococcus-pneumoniae* but they are asked separately with three intervening questions pertaining to *Hemophilus-influenzae*, another

organism. Thus, in pursuing a goal, MYCIN's reasoning is guided by a rule order which is somewhat arbitrary and unfocused as to the possible values for the goal.

A shortcoming of the implementation of MYCIN was that its explanations included a fair amount of extraneous bookkeeping information and required its users to be familiar with the details of the representation of its rules in the knowledge base. As a consequence, ordinary users and even computationally naive domain experts could not always follow its reasoning. This criticism is made even more clear in Figure 2-1, where the user dialog is placed at the center of the current assertions and the explanation capability.

Figure 2-3: RATIONALE's Integrated Explanation Capability

Therefore, users were expected to track MYCIN's 'state of the world' to be able to understand its explanations. Such a strategy is becoming less and less acceptable in the

design of human-machine dialog, where the users are deliberately being placed outside the application to shield them from the internals of the application[Hayes83]. In RATIONALE, as illustrated in Figure 2-3, the explanation capability is integrated with both the reasoning capability as well as the knowledge of the system. As a result of this integration, user interaction is performed through a single unified interface which serves as a shield from the internals of the application.

Despite its deficiencies however, MYCIN's explanation facility has served as a model for explanation in knowledge-based systems for well over a decade and its explanation capability remains better than many others.

2.2.2 GUIDON and NEOMYCIN

MYCIN's shortcomings were somewhat alleviated with GUIDON [Clancey86], a successor to MYCIN, intended as a system for tutoring medical students. MYCIN was first generalized to EMYCIN. GUIDON was designed to be domain-independent and as a result work with any EMYCIN knowledge base. NEOMYCIN, a successor medical diagnosis program, expanded MYCIN's disease knowledge to include competing alternatives. This expansion provided the opportunity for diagnostic strategy tutoring. The purpose of NEOMYCIN was to develop a knowledge base that facilitates recognizing and explaining diagnostic strategies. It resulted from GUIDON's requirements in extending MYCIN to tutor. This required that the former's reasoning strategies be separable from the domain and be explicit enough for explanation. GUIDON uses an 'overlay model', in which the student's knowledge is modelled as a subset of what the expert knows. It has four user levels ranging from beginner to expert with enhanced mechanisms for user tutoring.

However, like MYCIN, GUIDON is not built on the basis of a causal domain model. As a result, user explanations merely describe actions that happen to have been taken by the system without indicating why they are reasonable.

NEOMYCIN accomplished its objectives of explaining strategies by representing the control knowledge strategies explicitly in metarules. Such a representation allows NEOMYCIN to give strategic explanations and make clear the plans and methods it uses to reach a goal[Hasling84]. Citing the specifics of a situation while keeping the strategy implicit is the first form of its action explanations. For example,

'I'm asking whether the patient is receiving any medications
in order to determine if she's receiving penicillin.'

NEOMYCIN makes the underlying strategy explicit in its second form of system action explanations. Continuing with the above example,

'I'm asking whether the patient is receiving any medications
because I'm interested in determining whether she's receiving
penicillin. *I ask a general question before a specific one when possible*'

NEOMYCIN is based on the hypothesis that an 'understander' needs to have an idea of the problem-solving process, as well as domain knowledge, in order to understand the solution or solve the problem himself [Hasling84]. Its strategy is structured in terms of tasks, metalevel goals, subgoals and rules, with the rules acting as methods for achieving the goals. The metarules invoke other tasks, ultimately invoking the base-level interpreter to pursue domain goals or apply domain rules. The task structure represents a general diagnostic problem solving method. Note that the tasks and metarules are all domain-independent designed to be portable to other domains.

An ordered collection of metarules constitutes a procedure for achieving a task. The metarule premise indicates when the metarule is applicable while its action indicates what should be performed once the premise is satisfied. The premise examines either the domain-dependent knowledge base or the current problem history for findings and hypotheses such as follow-up questions for a recent finding or a subgoal of an active hypothesis. Associated actions would be to ask the user a question or call a task to refine the hypothesis under consideration. An example consultation session which illustrates NEOMYCIN's strategic explanations proceeds as follows,

Please enter information about the patient.

Name	Age	Sex	Race
1)**Mary	42	F	LATINO

Please describe the chief complaints:

****HEADACHE**
****STIFF-NECK**
****NAUSEA**

2) How many days has Mary's headache lasted?

****10**

3) How severe is Mary's headache (on a scale of 0 to 4 with 0 for very mild and 4 for very severe)?

****WHY**

[ie. why is it important to determine the severity of Mary's headache?]

[1.0] We are trying to characterize the data just obtained and develop the differential.

Headache severity is a characterizing feature of a headache.
 [back to question 3 ...]

To explain why NEOMYCIN is performing an action, the question is restated as 'why are we trying to do this task?'. The query is answered by stating the task and the metarule that invoked the task in question since at the lowest level, there are implicit tasks

of requesting data and applying rules. The user is given the option to ask for an abstract explanation as follows:

****ABSTRACT**

[ie. why is it important to determine the severity of Mary's headache?

[1.0] We are trying to characterize the data just obtained and develop the differential.

A datum that can be requested is a characterizing feature of the recent finding that is currently being considered.

When subsequent *why*'s are input, the strategy tree is ascended much as MYCIN's history tree was ascended. *How* is asked in terms of previously mentioned tasks. The explanation includes any completed metarules associated with the current instance of the task as well as the current metarule being executed. *How* explanations state the action of the metarule task whereas *why* explanations state the premise of the metarule task. An example of a *how* question at a low level in the strategy tree proceeds as follows:

****HOW 19.0**

[ie. Question 19.0: how are we trying to decide whether Mary has tuberculosis?]

We did the following:

[20.1] Applied a rule mentioning a causal precursor to tuberculosis (tb risk factor; rule 525), and

[20.2] Gave thorough consideration to any recent findings.

[20.3] Applied a rule mentioning general evidence for tuberculosis (focal neurological signs; rule 366), and

[20.4] Gave thorough consideration to any recent findings.

[20.5] Applied a rule mentioning general evidence for tuberculosis (compromised host; rule 309),
and

[20.6] Gave thorough consideration to any recent findings.

We are now:

[20.7] Applying a rule mentioning general evidence for tuberculosis (a history of granulomas on biopsy of the liver, lymph node, or other organ; rule 002).

We will next:

[20.8] Give thorough consideration to any recent findings.

[back to question 14 ...]

How questions can also be asked at an abstract level in order to inform the user of the control strategy that is being followed. For example,

17) Has Mary been hospitalized recently?

****WHY**

****HOW 25.0**

[ie. Question 25.0: how are we trying to develop the differential using basic history and pertinent physical exam?]

We did the following:

[26.1] Got a general idea of the problem: categorized it into one of several pathogenic classes or disease loci, or both.

[26.2] Confirmed and refined the differential diagnosis through specific questions.

We are now:

[26.3] Rounding out the diagnostic information by looking generally into past medical history and by reviewing systems.

[back to question 17 ...]

****NO**

NEOMYCIN uses explanation in a manner similar to MYCIN's. The main difference is that in MYCIN, rules invoke subgoals through their premises, while in NEOMYCIN metarules invoke subtasks through their actions. This results in the latter explaining at the level of general strategies instantiated with domain knowledge, when possible, to make them concrete. NEOMYCIN shares some of MYCIN's shortcomings in explanation. Its explanation detail includes all the information associated with every task

and metarule. Such detail will likely overwhelm users with bookkeeping information as MYCIN explanations do.

Another NEOMYCIN shortcoming is that it does not explain why metarules fail, only why they succeed. The ability to explain a failed meta-rule is important to users who are not familiar with the system's reasoning strategies as well as to domain experts evaluating system failures as well as successes. Such knowledge illustrates errors in system reasoning and deficiencies in knowledge compilation. The explanation mechanism for *why* a meta-rule failed is similar to explaining *why* a meta-rule succeeded. In the former case the system would simply inform the user of the failed premises or conclusions of the invalid meta-rule. This mechanism is described in Chapter 4; it is used in RATIONALE to explain *why* the system failed to make a deduction expected by the user.

Clancey's work with NEOMYCIN and GUIDON has evolved into three subsystems. The first is GUIDON-WATCH, a graphic interface to NEOMYCIN that uses multiple windows to allow the user to browse through the knowledge base and view reasoning processes during a consultation. The second is GUIDON-DEBUG, a system that allows the developer to roll back the consultation display and view the windows of the reasoning processes at any time a question was asked. This enables the users to trace changes in system values and resultant inferences.

GUIDON-MANAGE is the third subsystem and it is planned as a tutoring aid for medical students. The premise for this system is that 'a diagnosis is not the name of a disease but an argument which causally relates the manifestations which need to be explained' [Clancey86]. This implies that diseases are viewed as processes and a diagnosis is a network that causally links manifestations and states to processes. A causal explanation

provides a case-specific, or situation-specific, model by applying the general concepts and links in a knowledge base.

Clancey likens the explanation structure to a geometric proof in that it must account for all findings as well as be coherent and consistent. The situation-specific model is represented as a connected graph with one process at the root. This representation was first illustrated in GUIDON-WATCH to provide a user with a graph that illustrates how the final diagnosis related to the known findings. This idea is the basis for making the instructional program, GUIDON-MANAGE.

The student carries out the diagnosis by posting hypotheses and linking them to known findings. As the student requests data, an operator checks if the requests are valid according to the evolving network links. Therefore, the student provides links that explain the findings that need to be explained and refine the hypotheses that need to be made more specific. Thus, the student 'manages' the diagnosis by explicitly applying strategic operators. Clancey does not describe the feedback that the student is given, but if one assumes that the GUIDON-WATCH windowing is carried over, the reasoning processes would be illustrated in windows.

A criticism that carries over from MYCIN to NEOMYCIN and to the GUIDON subsystems is the level and nature of detail that users are presented with. No strict distinction is made between users and developers. For example in GUIDON-WATCH, it is assumed that the users require the details of the reasoning processes that developers require. Generally, such an assumption is not true, developers require debugging details such as those in GUIDON-DEBUG, whereas end users require explanations for system reasoning and these should not simply be graphs of reasoning hierarchies. Such graphical

hierarchies would appear unsuitable for less experienced users and may only be useful for expert users and system developers who are helping to checking knowledge consistency.

2.3 Reasoning Trace and Deep Explanations

The third class of explanations is directly related to the reasoning taking place in the session. 'Reasoning' is not intended to simply mean pattern-matching and firing If-Then-rules that directly compile an expert's knowledge. Rather, 'reasoning' refers to the system's activity of accessing both a causal domain model and explicitly represented strategic or heuristic advice. The domain model is captured by a hierarchy of hypotheses which are refined by applying heuristics to observed symptoms. Separating domain knowledge from strategic knowledge has several well known advantages such as enhancing the system's modifiability. However, from RATIONALE's perspective, the greatest advantage lies in the fact that control decisions are not buried in the knowledge base, i.e. they are not hidden by details of the ordering of rules or of the premisses within rules, but are immediately accessible to the explanation generating module. As a result, genuine explanations can be given: instead of regurgitating the actions actually taken during a session, an explanation can indicate why an action is reasonable in the light of available symptoms, heuristics and domain principles, and why, for instance, an alternative action was not performed.

User questions in this form of reasoning, may be recognized by pattern matching, compared to system templates, parsed as natural language queries or simply selected from menus. Once a question is understood by the system, the reasoning trace is examined and

trace specialists generate text. This text may be produced by a natural language generator, a template-based text generator or consist simply of canned strings. What matters is not the sophistication of the text generator, but that explanations be based on the reasoning with hypotheses (in the above sense) that took place. To keep the human-machine interface simple, the current implementation of RATIONALE uses template-based text. The difference between such template text and that attached to rules is that the text associated with hypothesis-based reasoning explains why hypotheses are triggered as well as how they relate to the other hypotheses in the reasoning chain. Such explanations - at a user-specified level of expertise - are illustrated in the discussion of RATIONALE in Chapters 3 and 4.

2.3.1 BLAH

Several systems have been constructed to explain on the basis of the reasoning trace. One such system that explains on the basis of a reasoning trace is BLAH by Weiner [Weiner80a]. BLAH constructs explanations that are modeled after psycholinguistic studies carried out with humans. These studies determined that people often explain decisions, choices and plans by emphasizing differences between alternatives rather than similarities. BLAH is a question-answering system whose domain is the filing of income tax returns. It uses a data base of assertions and justifications to allow it to handle three types of questions. Users can ask it whether or not some assertion is believed. The form of such a question is (SHOW <assertion>).

For example, to find out if Herby had to file a tax return:

user: (SHOW (HERBY MUST FILE TAX RETURN))

BLAH: (HERBY MUST FILE TAX RETURN)
or (NOT (HERBY MUST FILE TAX RETURN)) ...the results of SHOW become assertions
with associated justifications

They can also ask BLAH to make a choice between two alternatives, the system resolves this by generating a hypothetical instance of each choice with justifications for its selection or rejection. This is similar to the system answering two what-if questions and interpreting its own answer. The command word for this type of question is CHOICE. For example, to find out if a long form or a short form should be used to file one's income tax return:

user: (CHOICE (USE LONG FORM)(USE SHORT FORM)())

BLAH: (I CHOSE (USE SHORT FORM))(NOT (USE LONG FORM))) ...the results of CHOICE become assertions with justifications that are used for further explanation as shown below

RATIONALE attempts to take these results of choices into account by providing hypothetical explanations which justify an event by pointing out why otherwise likely alternatives did not occur.

Users can ask BLAH to explain why some assertion, already in the knowledge base is believed. The form of such a question is (EXPLAIN <assertion>).

Continuing with the previous example, if one wants to know why HERBY does not have to file a tax return:

user: (EXPLAIN (NOT (HERBY MUST FILE TAX RETURN)))

BLAH: Herby does not have to file his tax return because he made less than 3000 dollars last year.

Note that the EXPLANATION command answers are generated using a template-based text generator and are less awkward than the previous two types of responses. However, all three types of explanations restrict the user to input a question format similar to a pattern-match program invocation call.

BLAH reasons to prove that an assertion is believable as well as track certain features it requires for generating an acceptable explanation [Weiner80a]. Its knowledge is organized into a reasoning component as well as an explanation generator as illustrated in Figure 2-4. The knowledge is represented as sets of production rules. Attached to each assertion may be several justifications each consisting of a justification type as well as two lists. The first list contains the set of assertions justifying belief in the assertion and the second, which is optional, contains those justifying disbelief. The knowledge base is segmented into worlds. A world consists of a set of unique initial premises. This allows BLAH to reason hypothetically by enabling it to create new worlds based on some hypothesis.

Figure 2-4: Organization of BLAH

The output of the reasoning component is a tree representing a statement and its justifications. The explanation generator translates the reasoning tree into text, which is generated using templates associated with each assertion. BLAH views explanation as a 'negotiation process', where the user is able to enter a series of commands requesting the system to explain in more detail aspects of previous explanations, given as a function of the depth of the reasoning tree.

BLAH compares favourably with other explanation facilities but its primary limitation is the restriction it places on the format of the user's question. It requires that the user type in a question in a highly artificial form which is very much like a pattern-directed program call. BLAH lacks meta-level knowledge structuring in its domain knowledge. It requires that the user be familiar with the data base structure and organization so as to supply a list of data base partitions, again in restrictive pattern-directed command syntax, to guide BLAH in its reasoning. Despite these shortcomings, BLAH focuses on one of the most important components of explanation, the user's view. It presents very useful alternatives to modeling, such as modeling a user's view versus the system's (expert's) view in explaining system reasoning.

2.3.2 XPLAIN and EES

The most ambitious system constructed to explain on the basis of a reasoning trace is Swartout's XPLAIN [Swartout81; Swartout83]. XPLAIN is based on two key principles: explicitly distinguishing different forms of domain knowledge represented in the knowledge base and formally recording the system development process. As an automatic programmer, the knowledge needed to write the expert system is available to the expert system itself in the form of a domain model and domain principles, as illustrated in Figure 2-5. The domain model is a network of causal relationships and the domain principles express reasoning strategies that guide the knowledge refinement process.

Figure 2-5: XPLAIN's knowledge Separation

As the system reasons, it constructs refinements which are then used for explanation. The explanation is based on justifications produced by refining the code. XPLAIN has been applied to the Digitalis Therapy Advisor which advises physicians on the level of the digitalis drug administered to congestive heart failure patients. With its knowledge separation, the system is able to provide justifications for executed rules, allow diagnostic methods to be stated more abstractly as well as allow the domain model and principles to be modified independently of one another.

The general structure of XPLAIN is illustrated in Figure 2-6. The knowledge needed for its automatic programmer to generate the code is abstracted through refinement by a hierarchical planner from the domain model and principles. The trace of writing the expert system using the generated code becomes the refinement structure. This structure documents all strategies taken in writing the code. The top refinement structure becomes the top goal and is refined into subgoals. The leaves of the refinement tree form the performance program. The program writer algorithm proceeds as follows:

```

choose a step to refine
check if there is a method for this step
  if true, proceed by creating a call to the planner
find a domain principle
check if the constants are valid
  if the constants are valid, instantiate a method
  if the constants are not valid, find another domain principle
create a call to the planner
add a new step to 'steps to refine'
remove step just refined
loop (... choose another step to refine)
  
```

Figure 2-6: General Structure of XPLAIN

The automatic programmer starts off with partial specifications that are expanded by matching against the domain model. The instantiated goals of a method are placed by the writer into the refinement structure, by the 'call to plan', as children of the goal being refined. XPLAIN collects the refinement structure as knowledge needed for explaining, synonymous with knowledge it requires to generate automatic code.

XPLAIN answers user questions about its abilities, the domain principles, its session reasoning as well as its static domain knowledge. It parses natural language questions and uses a language generator to output the explanations. It generates answers using an English phrase generator which converts the refinement structures into English templates. Its explanations are sensitive to the current context the user is in, knowledge of what it has already explained as well as a model of the user to predict the level of detail of the explanation.

For example,

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

User: why?

XPLAIN: The system is anticipating digitalis toxicity. Cardiomyopathy causes increased automaticity, which may cause 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.

The explanation is quite explicit, somewhat detailed, and is of high quality English as many users desire it to be. An example of a domain principle explanation is as follows:

User: Describe how digitalis toxicity can be anticipated?

XPLAIN: 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.

XPLAIN also describes its prescriptive knowledge in answering *how* questions:

User: How did the system determine that serum calcium was increased?

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

XPLAIN demonstrates a powerful concept in expert system development, but it is not without limitations (see [Neches85]). Although one of the primary objectives of XPLAIN is to use domain justifications to indicate correspondences between terms at different levels of refinement, it uses terminology inefficiently. Due to a lack of explicit term definitions in the domain, term definitions are dependent on the context the program writer is in. Therefore, a term definition provided by a domain principle is uniquely associated with that specific piece of problem solving knowledge. Such a strategy restricts term definition knowledge from being shared across individual domain principles.

Another shortcoming of XPLAIN is the restriction placed on the explanation generator. It consists of fixed procedures capable of answering a limited number of question types. Thus, the system can only answer a very limited set of questions quite well. Limiting explanations as such illustrates that Swartout's facility is similar to those explanation facilities which are not fully integrated with the system and as a result they track rather than explain reasoning. Swartout's published results are misleading in this respect since he writes without informing the reader that the explanation generator is restricted [Swartout83] and that his approach has been successful for a limited set of explanations. Another limitation of XPLAIN is that its automatic program writer is restricted to predefined goal/subgoal refinement. Thus, if no defined principle could be found to refine a goal, the system stops without explanation.

Swartout and Neches [Neches85] are in the process of defining EES, the Explainable Expert Systems approach, illustrated in Figure 2-7. EES is directed at two tasks: the task of generating explanations to clarify or justify the behaviour and conclusions of the system; and the task of extending or modifying the system's knowledge base or capabilities. This system is their response to the shortcomings of XPLAIN. In many ways

EES is similar to RATIONALE, however, the builders of EES are using the automatic programmer to capture the knowledge of the human programmer. The use of the automatic programmer adds a level of complexity to the expert system explanations which as RATIONALE illustrates, is not necessary. It is possible to attain EES's goals of explanation by encoding the human's reasoning in the system much as the explicit reasoning tracking mechanisms in RATIONALE do. Thus, the program writer function is replaced in RATIONALE by explicit reasoning mechanisms that are encoded by the programmer to provide explanations at the same level as EES is striving for.

EES, like RATIONALE, aims to shift the knowledge engineer's emphasis from procedural encoding to declarative knowledge representation. With collaboration from domain experts, the plan in EES is to generate a knowledge base augmented with implementation strategy advice. The knowledge base would then be used by the automatic program writer to generate the code for the expert system while maintaining a trace of 'development history'. The code executed by the interpreter would then become the 'execution history' to be used for justification and explanation. According to the developers, the automatic programmer is essential for simplifying the task of expert system maintenance. Such a claim is debatable. In RATIONALE, as in EES, high-level strategies are easily expressed, classifying knowledge makes the system modular and new knowledge is easily classified, therefore making the system easily modifiable without the existence of an automatic programmer.

The domain model in EES (as in XPLAIN) will describe how the domain works by using typological and causal links. Domain principles represent the problem solving strategies and are used by the program writer to drive the refinement process. They associate tradeoffs with domain principles to indicate the advantages and disadvantages of a

choice of strategy. Similarly, preferences are associated with goals to set priorities based on tradeoffs. EES is planned to address XPLAIN's terms definition limitation by separating the definition of terms from the contexts that use them. Integration knowledge will serve to resolve potential conflicts among knowledge sources as a basis for a single recommendation. Optimization knowledge is intended to direct the execution of the expert system in the most efficient way given a particular context. The knowledge base will be represented in NIKL, a refinement of KL-ONE [Neches85], a semantic network-based representational formalism. Given a new concept to add to an existing network, the NIKL classifier automatically places the concept in the subsumption hierarchy of the network.

Figure 2-7: Planned Structure of EES

EES will generate a runnable expert system by applying the program writer to the knowledge base. The steps taken in generating the code record the development history (or

refinement structure as defined previously in XPLAIN). While refining EES will allow goal/subgoal refinement as in XPLAIN, *Covering Reformulation* or generating a subgoal hierarchy from the original goal to cover the possibilities presented by the original goal is a planned enhancement. As stated earlier, a limitation of XPLAIN is that its automatic program writer is restricted to predefined goal/subgoal refinement. If no principle could be found to refine a goal, the system stops. Such a restriction reduces the reusability of the system's problem solving knowledge in new situations.

In EES, the program writer is being defined to be capable of reformulating a goal when a match cannot be found. The reformulation is intended to effect a more explicit development history which is expected to aid explanation. Note that RATIONALE also makes use of similar goal/subgoal refinement without automatic programming. Again, it is not clear why it would be useful to complicate reasoning and explaining by introducing automatic programming. In RATIONALE, the function of the automatic programmer, that of setting up the system's goals, is identical to that of the human programmer initially defining the expert system domain. The programmer is given a flexible environment in which to define the goals and their attributes, thus, the system will refine those specified goals, rather than generating its own set as both XPLAIN and EES do.

User directed dynamic refinement is another form of refinement aimed at allowing the user to select a refinement strategy for the system to pursue. This is a feature already implemented in RATIONALE which allows the user to direct the strategy of refinement. *Input-Based* reformulation is another refinement option in EES allowing access to a specialized plan at run-time to handle a goal that has a very general input that cannot be handled by any of the available plans.

The EES interpreter executes the leaf nodes of the development history to produce an execution trace and it also manages the system's normal interaction with the user. User questions are directed to the explanation generator, which accesses the knowledge base, development history, interpreter, and execution trace to construct responses.

XPLAIN's explanation generator is limited to answer only a few types of questions. In addition, the answer generator consists of fixed procedures coded into XPLAIN itself. EES is viewing explanation as a planning task while expressing knowledge as a set of declarative explanation strategies. As in RATIONALE, EES will classify user questions into question types about system behaviour such as justification and appropriateness of system action, as well as definition of, and capabilities in, system concepts. Unlike RATIONALE however, EES is not addressing 'what-if' or hypothetical explanations.

2.3.3 MDX, PATREC and CSRL

The approach to reasoning in RATIONALE has been influenced most strongly by Chandrasekaran, in particular, the reasoning in his system MDX [Chandrasekaran: 79; 82; and 83], its knowledge directed database, PATREC [Mittal84], and the classification language CSRL [Bylander86].

In early work on MDX, Chandrasekaran proposes 'conceptual structures' as the knowledge representation scheme [Chandrasekaran79]. He defines concepts as labels which organize how-type or methodology knowledge, by representing them as a collection of diagnostic production rules [Gomez81]. The intent of this early scheme is to group diagnostic rules into an accessible knowledge structure, with this knowledge structure

evolving into a frame in later work on MDX [Chandrasekaran86a]. The structure is dominantly a hierarchical one, with the successors of a conceptual node representing subconcepts which refine that node. Associated with each concept is a set of procedural 'experts' which attempt to apply the relevant knowledge to the current case. The procedural experts also have the ability to relinquish control to selected subconcepts for more detailed analysis. The conceptual structure can be viewed as a means of organizing knowledge so as to allow purposeful, focused access to concepts relevant to a particular context in a user session. The problem solving strategy is one of hypothesize and test: given a set of user symptoms, generate hypotheses that match the symptoms and test whether these hypotheses give a valid diagnosis.

Chandrasekaran draws a relationship between the principles that lead to effective communication amongst physicians and those used to organize and structure knowledge within a medical concept hierarchy. One of these stipulates that there exists a hierarchical organization in which control transfer is achieved (eg. between a General Practitioner and a Specialist). Also, the expert contemplating control transfer is knowledgeable enough of the domain subexperts to know which are relevant to a particular problem. The decision to consult a particular subexpert is based on the observed symptom data and how well it matches a subexpert's knowledge. Moreover, each expert is knowledgeable enough about his domain of specialty to conclude when he has been mistakenly called. RATIONALE takes this one step further by encoding the knowledge to suggest immediate alternate hypotheses in the expert's domain knowledge. These were some of the main principles applied to organize the knowledge for MDX, a system that diagnoses liver diseases or Cholestasis. Its conceptual structure, illustrated in Figures 2-8 and 2-9, forms a tree, where a node represents a concept and its refinements represent its subconcepts. Chandrasekaran

views the principles of organization of medical knowledge to arrive at a solution, as taxonomic classification. He defines the problem as that of identifying the concept's own superordinate and subordinate concepts. This is a process of recognition that is intrinsically top-down. Note that the principles of organization of conceptual structures use knowledge efficiently as well as guide in knowledge acquisition. Also, heuristics that enable a system, such as MYCIN for example, to proceed to a concept several levels below, sidestepping the methodical level by level communication, obscure the clarity of the underlying structure [Chandrasekaran79].

Figure 2-8: Conceptual Structure of Cholestasis

Cholestasis is a condition caused by disruption of the secretion of bile or of its excretion into the duodenum. The causes of the disease are grouped according to whether they occur within the liver (intra-hepatic) or in the biliary tree outside the liver (extra-hepatic). For extra-hepatic causes, the movement of bile through the biliary tree can be

obstructed by a physical object, such as a stone, stricture, or various forms of cancer, or by inflammation swelling due to a number of causes. Each of the subconcepts has an associated set of symptoms and procedural methods for establishing them. Because of the attached procedures, the nodes of the tree can be viewed as a structural organization of experts involved in the diagnosis of Cholestasis [Chandrasekaran79; Chandrasekaran81].

Figure 2-9: Remainder of the Conceptual Structure of Cholestasis

The experts are categorized according to their tasks. The top level concepts such as Cholestasis, Extra-hep and Intra-hep, control the diagnosis decisions. They accomplish this by directing the system to the experts to call next, the subproblems to solve, as well as mediating between the competing advice between subconcepts. The other class of experts such as stone, cancer and stricture evaluate the problem data by matching to determine how well the concepts fit the data they represent and what additional data will aid the decision. This class of experts are considered tip nodes or primitive concepts since they match the

observed symptoms to the symptoms required to validate the hypothesis, without interacting with other hypotheses in the tree. The problem solving strategy is one of hypothesize and test. Some nonprimitive concepts are 'anchoring' concepts. They can be established with a high degree of certainty. Once established, they provide a strong global focus to the problem-solving process and they distribute control to the other experts so that they can be further refined. A condensed example of Cholestasis diagnosis by MDX:

PATIENT RECORD SYSTEM

PATIENTS NUMBER IS PAT0

AGE? 39

SEX? M

ENTER HISTORICAL EVIDENCE

? HALOTHANE

? (ULC-COLITIS F)

? (BIL-SURGERY F)

? (CHOLECYSTITIS F)

? E

TIME? (AT ADMISSION)

? JAUNDICE

? PRURITUS

? VOMITTING

? (WT-LOSS (AMOUNT (LB 16.0)))

? (PAIN F ABDOMEN)

? (LIVER (SIZE NORMAL))

? (PAIN LIVER F)

? (PAIN T EPIGASTRIUM)

? (SPLEEN (FELT F))

? (URINE (BILIRUBIN))

? (STOOL (BLOOD F))

. the data entry continues with a similar dialog of pattern invocation-like format

. **** note: F in the above dialog implies FALSE

END OF DATA ENTRY FOR PATIENT PAT01

BEGIN DIAGNOSIS FOR PATIENT PAT01

TRYING RULES TO INVOKE HYPOTHESIS

TRYING RULE:

((((PRESENT? (URINE BILIRUBIN))(NORM? ALK-PHOS AN))
((INVOKE CHOLESTASIS)))

SATISFIED

EXECUTING ACTIONS: ((INVOKE CHOLESTASIS))

TRYING RULE:

((((HISTORY? ANESTHETICS))((INVOKE (POST-OPER ANESTHETICS)))))

SATISFIED

EXECUTING ACTIONS: ((INVOKE (POST-OPER ANESTHETICS)))

TRYING RULE:

((((DEFORMITY? BIL-TREE ? STONE))((INVOKE STONE)))

UNSATISFIED

INITIAL DATA INDICATES - - -

CHOLESTASIS

(POST-OPER ANESTHETICS)

CHOLESTASIS IS INDICATED BY

PRURITUS PRESENT

JAUNDICE PRESENT

URINE BILIRUBIN

the concluded facts about Cholestasis are listed

(POST-OPER ANESTHETICS) IS INDICATED BY
HISTORY OF ANESTHETICS

HANDING OVER CONTROL TO CHOLESTASIS EXPERT

EXPERT-CALLED: CHOLESTASIS

CALLED-BY: GP

MESSAGE (QUERY (ESTABLISH CHOLESTASIS)(REFINE CHOLESTASIS))

TRYING TO ESTABLISH CHOLESTASIS

ENTERING TABLE LOGIC

TRYING RULE: (((DUCT? IHD BIOPSY BILE-STASIS) (PRESENT? JAUNDICE)) ((T 3)))

UNSATISFIED

EXITING TABLE LOGIC

PATHOGNOMONIC EVIDENCE IS 0

PHYSICAL EVIDENCE 4

CLINICAL EVIDENCE (8 4 0 2)

CHOLESTASIS FIT: (ESTABLISHED CHOLESTASIS 3)

From the above, one notes that data is input from the user in a *data entry phase*.
Once the data entry is complete, the top level diagnosis expert, GP, fires off rules to

suggest starting level hypotheses. In the above example, the Cholestasis and Post-operative-cholestasis experts are suggested by the rules. Cholestasis being the highest anchor node suggested implies that control is transferred to it with the message to establish and refine itself. Cholestasis establishes itself with an evidence of +3 from a table with scales of +3 to -3. Once Cholestasis is established, control is passed to Post-operative-cholestasis to establish itself. In this example the latter was not established. The refinement continues with Cholestasis trying to establish extra-hepatic and intra-hepatic causes by refining its subconcepts. Once the diagnosis is complete, a list of considered diseases is summarized, each ranked according to the evidence gathered.

MDX has no global calculus for handling uncertainty [Chandrasekaran82], as offered by Bayesian techniques, or as there is in MYCIN. This is a consequence of having no separation between the knowledge base and the problem solver. The diagnostic specialists represented as concepts in the hierarchy work under the hypothesize and test strategy, hence, they hold distributed knowledge and represent distributed problem solvers. This approach provides the flexibility to make each specialist combine the uncertainties of its constituent data or knowledge in a manner appropriate to the context [Chandrasekaran86c]. Most medical decision-making systems are faced with the task of reasoning from uncertain knowledge of various types and arriving at a credible definitive decision. MDX uses an uncertainty mechanism in which the underlying knowledge that drives the diagnosis is represented by two collections of numbers which denote likelihood relations between manifestations and diseases. These numbers may be construed as probabilities but their combination during problem solving is not Bayesian.

Figure 2-10: Collecting Evidence in MDX

MDX uses a number of AI techniques such as focusing mechanisms and hierarchical reasoning to avoid the problems associated with Bayesian computations. MYCIN's knowledge representation is in the form of rules which relate a set of premises to a possible conclusion and associate with it a 'certainty factor'. Its problem solving mechanism is one which propagates conclusions with varying degrees of certainty: weak ones below a certain threshold are dropped while strong ones above the threshold are kept.

MDX does not handle uncertainty in an absolute numerical scale, rather it makes decisions on a discrete scale of a few distinct values as detailed below. Therefore, MDX

uses a framework which handles uncertainty in a hypothesis-specific manner. It has no explicit rules of combination, rather it makes use of tables of hypotheses and evidence explicitly filled out by human experts. Consider the evidence collection for establishing 'stone', a tip node in the conceptual hierarchy. Assuming that the ancestral specialists of stone are established to a required degree of certainty, consider its contributing concepts in Figure 2-10. In Figure 2-10 (b), the numbers in the 'clinical evidence' column are not constructed by any formula, global or local. They are filled according to the knowledge of human experts as follows:

given reasonable evidence for cholestasis, in particular extra-hepatic obstruction, and given that the patient has cholangitis and colicky pain, but no vomiting or nausea, how much weight, in a scale of +3 to -3, will you give for clinical evidence for biliary stone?

Once values for the clinical and historical evidences are synthesized, the process is repeated for the next higher level as illustrated in Figure 2-10 (c). This manner of handling uncertainty appears to be a promising one and is similar to the one included in the overall framework of RATIONALE. In RATIONALE, a hypothesis is satisfied by successfully matching the observed symptoms to those the hierarchy specialist expects according to a degree of match that the knowledge engineer sets. For example, if the degree of match to refine a particular hypothesis is 50%, then at least 50% of the symptoms required to refine the hypothesis must be observed. The element of uncertainty is introduced in the hypothesis within the affected context. If the hypothesis is refined, the new uncertainty would be that of the next hypothesis to refine with a degree of match independent of that of its parent. Thus, like MDX, RATIONALE handles uncertainty on a discrete scale of distinct values that are hypothesis-specific.

MDX is one of the first successful diagnostic expert systems to be implemented with the notion of conceptual hierarchies. It uses its knowledge in context, once a context is established, it can reason about data at the appropriate level in the concept hierarchy. However, it lacks a good user dialog mechanism. User 'explanations' are given as a programmer trace output, while user input is like that in BLAH, pattern-invoked program input.

PATREC is a knowledge-directed database [Mittal84], that has been developed in conjunction with MDX to address amongst other things, issues in the following example [Chandrasekaran86a]:

Consider a problem in classificatory problem solving, there is some knowledge in the liver specialist that states, 'if history of anesthetic exposure, consider hepatitis'. If there is no mention of anesthetics in the patient record even though the patient has had major surgery, the competent physician will infer possible exposure to anesthetics, and consider hepatitis.

Mittal [Mittal84] notes that this inference did not require classificatory reasoning, rather it required 'knowledge-directed information passing'. Knowledge about each data concept, not the data values for a particular patient, rather, domain knowledge such as default values for attributes and strategies for inferring them, is stored in a frame, with the frames organized into a frame hierarchy. Reasoning here involves accessing a frame that stores either the desired datum, or information about how the value of the datum might be obtained (including default values).

Continuing with the above example leads to the following:

The frame associated with anesthetics will examine its portion of the database, find no patient datum in it, and fall back onto the following knowledge: a) check if particular types of anesthetics were administered; b) check if any major surgery was performed and if true, answer 'yes'; c) if answer is 'no' by any of the above inferences, default to 'no'.

As no record of anesthetics will be found, it will check the 'surgery' frame and eventually infer that it had been performed. The anesthetic concept can then infer that the patient had been exposed to anesthetics and therefore, may have hepatitis.

PATREC has very powerful knowledge structures useful for building expert systems that reason by explaining. However, it uses the same awkward dialog format of MDX, illustrated previously. CSRL (Conceptual Structures Representation Language) is another resultant building block from the MDX work [Bylander86]. It is a programming language intended to facilitate the construction of knowledge-based systems that combine classificatory reasoning with uncertainty handling.

CSRL generalizes the diagnostic problem solving of MDX. It implements knowledge-based systems as a classification hierarchy of specialists intended to be used for hypothesis refinement. It uses the top-down control strategy of establish-refine or hypothesize and test, in which either a specialist establishes and then refines itself or rejects itself, thus pruning the hierarchy it heads. The most common approach to building knowledge-based systems is to adopt a general knowledge representation, such as MYCIN's production rules, and a separate inference engine for using that knowledge. The assumed advantage of this approach is that *knowledge* is separated from its *use* and consequently, incorporating *new knowledge* is just a matter of *adding new rules*. In some cases, this approach may be better than the standard systems development approach which required that an explicit algorithm be formulated to embody *how* to accomplish a particular

task. Realistically however, this approach has severe limitations and is not useful for solving more complex tasks such as those in medical diagnosis.

This is why the underlying philosophy of CSRL is that knowledge cannot be effectively separated from its use [Dudzinski86]. CSRL has a number of elementary organizational and information processing strategies, known as generic tasks [Chandrasekaran86a], which can be used as building blocks in modeling expert problem-solving behaviour. Classification is one of these generic tasks. CSRL is intended to facilitate knowledge acquisition and enforce consistency in knowledge use and representation, by allowing medical classification knowledge to be captured as hierarchies of knowledge and methods specialists, similar to its existence in a physician's conceptual level.

RED [Josephson84], an MDX successor, is an expert system whose domain is red blood cell antibody identification, for use by physicians to decide on the match between blood used for transfusions and blood for patients undergoing surgery [Bylander86]. CSRL is used to implement specialists corresponding to each antibody that RED knows about (approximately 30 types) and to each antibody subtype (different ways in which the antibody can react). Other expert systems that CSRL is a part of are [Dudzinski86]: WELDEX, a diagnostic system intended for the inspection of radiograph weld images and the identification of faults in welded pipes; ROMAD, a system for the diagnosis of problems in rotating machines; COREX, a prototype for diagnosing corrosion problems in mechanical systems; and AUTOMECH, a system that diagnoses fuel problems in automobile engines [Bylander86]. All CSRL-based systems share MDX's knowledge organization and representation framework of conceptual structures. This approach is being proven as a promising one for building diagnosis systems that reason hierarchically. In all

these systems however, the explanation of system reasoning ranges from poor to non-existent [Dudzinski86].

Explanation of system reasoning seems to have taken a secondary role in Chandrasekaran's work. The term 'explanation' has been used as meaning reasoning by hypotheses that 'explain' the symptoms, rather than informing users of the strategies that the system follows as well as answering their questions about reasoning as RATIONALE, XPLAIN and BLAH attempt to. However, Chandrasekaran presents a powerful framework for reasoning explicitly for the purpose of explanation which has been key in developing the framework for reasoning in RATIONALE.

As in MDX and its successors, reasoning in RATIONALE follows a hypothesis tree. The hypothesis tree is built from the top down, i.e. from the most general diagnosis of a problem to the most specific. As the user inputs more symptoms for a diagnosis, a focusing algorithm guides the refinement of the hypothesis tree. RATIONALE's explanations of the reasoning trace are directly related to triggered hypotheses. These are used to explain to the user the validity of the hypothesis and any related refinements. Explanations of session events and abilities aimed to answer user questions about current reasoning are given on the basis of the generated hypothesis trace. These are elaborated upon in the next chapter.

The key difference between RATIONALE's approach and Chandrasekaran's is that in RATIONALE, the diagnosis requires that the system reason explicitly so that it may explain to itself, and if asked, explain to the user, its decisions in reasoning to refine any hypotheses. Thus, all validating and invalidating conditions for hypothesis refinement as well as alternative points of diagnosis on the hypothesis tree are tracked to allow the system

to reason explicitly and as a result explain. Although MDX and its successors have the framework to provide such explanations, they do not as of yet.

The reasoning trace of the executed program seems to provide a very promising basis for constructing helpful, genuine explanations. As Chandrasekaran notes [Chandrasekaran86a], reasoning in order to explain hypotheses, directly affects the processes of knowledge acquisition and domain modeling. The assumption of the approach in RATIONALE is that building an expert system is inseparable from building an explanation facility. After all, a knowledge-based system can explain its reasoning only to the extent to which it has access to a sufficiently rich record of its reasoning.

2.4 Adequacy Criteria for Knowledge-Based Explanations

To adequately summarize the key points in current explanation facilities, one needs to consider a set of adequacy criteria. The criteria fall into the six categories described below.

Knowledge Organization

It is important to recognize at this stage that building 'deep explanation' facilities is directly tied in to developing an expert system in the first place. Static knowledge to answer simple questions about facts in the system and dynamic knowledge for explaining the historical reasoning in a session should be easily accessible. In systems such as XPLAIN and EES, knowledge is organized into descriptive and prescriptive knowledge, as summarized in the table of Figure 2-11.

The prescriptive knowledge is used to reason about the descriptive knowledge. In MDX and RATIONALE, both types of knowledge are organized into hypotheses or concepts to keep knowledge relevant to a particular aspect of the domain together. Knowledge in the above systems is organized in a well structured manner, unlike MYCIN and its successors.

Knowledge in MYCIN is not structured, a single set of linear production rules are used for reasoning, thus restricting the value of explanation. Its knowledge representation is *flat*, it has no explicit structures or hierarchies that relate the various components in its

knowledge base. Much of the information required to formulate MYCIN's rules, including justification, ordering, and planning, is either lost or made implicit and thus, without a domain model, it is impossible to explain. Similarly, NEOMYCIN uses meta-rules which have been criticized as *ad-hoc* solutions to the problems of domain modeling [Gomez81].

Knowledge of the domain model gives the expert system the necessary depth to reason about its actions. Chandrasekaran states that not all the deep knowledge may be required for a particular problem of diagnosis[Chandrasekaran83], but it is certainly required for good explanations. This is debatable, RATIONALE uses deep explanations to aid its reasoning. Its approach is intended to be extendable in the future to reasoning about difficult problems by using deep explanations that it can incorporate into its diagnoses. Shallow reasoning in flat knowledge representations, as in MYCIN, often presupposes knowledge of why a hypothesis was triggered or refined into a more specific hypothesis. Thus, for good explanations, knowledge should be organized into hypotheses or structures that can properly represent the diagnostic domain.

	<u>Knowledge and Organization</u>	<u>Knowledge Representation</u>	<u>Explicitness of Mechanism Tracking Reasoning</u>	<u>Question Types*</u>	<u>Method of Explanation</u>
MYCIN	rules	rules	none (Program Trace)	Event, Ability, Factual	templates, canned text
NEOMYCIN	meta-rules	meta-rules	meta-rule (Program Trace)	MYCIN's	MYCIN's
GUIDON	meta-rules	meta-rules	MYCIN's	MYCIN's	MYCIN's
BLAH	worlds =>	rules	Reasoning Trace of Assertions & Belief/ Disbelief Justifications	Hypothetical, Event, Factual	templates
XPLAIN	descriptive => & prescriptive knowledge	semantic network	Refinement Trace of Automatic Program Hierarchy of Goals	Event, Ability, Factual	explanation generator
EES	descriptive => & prescriptive	semantic network knowledge	XPLAIN's	Event, Ability, Factual	explanation generator
MDX	conceptual => hierarchies	frames	none published	none published	none published
RATIONALE	conceptual => hierarchies	frames Enabling/	Reasoning Trace of Ability, Invalidating/ Special Case/ Refinement/ Alternate Conditions	Hypothetical, Event, Factual	templates

=> represented by

* refer to section 3.5.2 for a full description of the question types

Figure 2-11: Comparison of Explanation Facility Adequacy Criteria

Knowledge Representation

Knowledge representation is another important adequacy criterion. It is closely related to the problem of knowledge organization, since one choice affects the other. In XPLAIN and EES, knowledge is represented using semantic networks. In MDX and RATIONALE, knowledge is represented as frame hierarchies of concepts or hypotheses. In MYCIN and BLAH knowledge is simply represented as production rules. In NEOMYCIN and GUIDON, knowledge is represented by meta-rules. Frame representations provide a system with a representation that groups together knowledge about its component objects and its ancestor-sibling relations. For the representation of knowledge to be adequate, a model of the domain with a clear history of the interactions between its components during reasoning needs to be accessible. This criterion is satisfied by both semantic networks and frame hierarchies.

Explicitness of Mechanism that Tracks the Reasoning

An important adequacy criterion is the explicitness of the mechanism which tracks the reasoning in the system. In MYCIN and its successors, there exists no explicit mechanism to track reasoning, explanations of system behaviour are generated using a program trace. However, in BLAH, XPLAIN, EES and RATIONALE, the mechanism generates a refined trace. The refined trace holds only the reasoning information relevant

for explanation. As illustrated in the previous section, the best explanations are those derived from the most explicit refined trace. BLAH uses a reasoning trace of assertions with belief and disbelief justifications that are attached to its production rules to explain its actions. XPLAIN and EES use a refinement trace generated by their respective automatic programmers to explain their hierarchical goals. Chandrasekaran has not yet published information about how explanations are generated in MDX. As illustrated in Figure 2-11, RATIONALE has the most explicit reasoning trace.

RATIONALE's refined trace includes very detailed knowledge of hypotheses refinement. Hypotheses are triggered by a set of enabling conditions which are checked against a set of invalidating conditions. If a hypothesis becomes invalid, a set of special case tolerance conditions are examined to override the invalidation. Once the hypothesis is valid again, it is refined. In addition, a set of conditions for an alternate hypothesis are checked, if one is enabled, a trace is kept of its relation to the original hypothesis and the alternate is later refined once the original hypothesis refinement is complete. Thus, its refinement trace holds information about enabled, invalidated, special case, alternate and refinement hypotheses. Such a rich reasoning trace is used to generate very powerful explanations.

Question Types

Another criterion is the adequacy of the question types the system handles. Figure 2-11 summarizes the types of questions that are handled by the systems discussed previously. Ideally, question types should include: hypothetical or what-if questions; event

or why and how questions; system ability questions; and factual questions (see section 3.5.2 for a full description of these question types and their use in RATIONALE). The types of questions that are to be explained directly impact the knowledge representation in the knowledge-based system. Rules are used in MYCIN for user questions that require matching on the premises or the conclusion of a rule. Frames and causal networks in RATIONALE and XPLAIN, respectively, are used for questions whose answers require knowledge of a structured domain model. The question types also impact the human-machine interface of the expert system. Dialog questions of unrestricted text require powerful natural language interfaces. Informing the user about the available types of explanations may be easiest in a menu-based or form-based interface. This criterion is further described in Chapter 3.

Method of Explanation Generation

Generated explanations are another adequacy criterion and they have taken many forms. The most common of which are canned text, templates, or natural language responses. MYCIN, BLAH and RATIONALE make use of templates, whereas XPLAIN and EES use natural language generation. Either method can be adequate depending on its implementation. However, natural language generation is complex and is likely to be restricted to the question types the system can handle, thus, it may be simpler to implement templates.

Adhering to the above adequacy criteria will result in a system that can explain its reasoning and it may also facilitate the process of knowledge acquisition by offering explanations even during the construction of a knowledge-based system.

In RATIONALE, menus were selected for the human-machine interface because they keep the interface straightforward and provide the user with a simple guide as to which explanations can be asked for at any point during a session. The knowledge representation is frame based to group together knowledge about the hypotheses as described in Chapter 3. The domain knowledge is represented in frame slots. Each hypothesis has three explanation slots which have attached strings of text appropriate for a particular user level. In addition, rules are attached to slots to facilitate access to knowledge relevant to particular hypotheses.

Chapter 3

RATIONALE

Objectives

There are two main objectives behind constructing a tool such as RATIONALE. The first is to provide an end user of a diagnostic expert system with a comprehensive explanation facility. The second objective is to construct an expert system tool that allows a domain to be modeled, modified and run as an application in a single integrated environment such as the one RATIONALE provides.

Approach

Josephson and Chandrasekaran in [Josephson 84] describe the process of knowledge-based reasoning about data which describes a particular situation and arriving at an explanatory hypothesis that best accounts for the data, as a process of *abductive inference*. To accomplish abductive inference it sometimes is necessary to assemble parts into a unified explanatory hypothesis. Such is the case in RATIONALE where, domain knowledge is organized into hypotheses and sub-hypotheses that can be explained by their

interactions. The interactions explain similarities, differences and alternatives between the various hypotheses. An explanation is assembled to account for given symptoms by refining top level hypotheses to their lowest level sub-hypotheses which explain the given symptoms.

The domain knowledge in RATIONALE is organized into a hierarchy of hypotheses represented by frames, as illustrated in Figure 3-1 (see implementation section 4.1 for the full hypothesis hierarchy of the example domain - the ACME workstation domain). At the root of the hierarchy is the top level goal or the overall aim of a particular expert system domain. RATIONALE could be used to define a multiple domain expert system with each domain having a unique aim and a unique hypothesis tree. The top level goal serves as an initial entry point to the domain under consideration. For example, if several expert system domains were defined in RATIONALE, eg. a heart, a liver and a kidney domain and the patient's symptoms related to a heart problem, then only the domain hierarchy for the heart problems would be searched for a hypothesis explanation.

Figure 3-1: Hypothesis Tree for ACME Workstation Example

The second level of the hypothesis tree classifies the problem under consideration into the top level subgoals of the domain. For a system that diagnoses problems with computers, such top level subgoals could be as general as whether the problem is a hardware, software or power problem. The user may focus the search of the reasoning mechanism by specifying at which subgoal the diagnosis should start. If the user has no idea, the hypothesis tree is traversed left-to-right, breadth-first and the user is questioned about observed symptoms until a match is found. In this case, the search is implicitly directed since the knowledge engineer was instructed by RATIONALE to define the hypotheses in decreasing order of priority, with the leftmost hypothesis having the highest priority. The top level subgoals are refined into more specific hypotheses on the basis of various relationships among the following sets of symptoms:

- S , the set of all possible symptoms that are relevant to hypothesis H
- $S_e \supseteq S$, S_e contains all symptoms that enable H
- $S_i \supset S$, S_i contains the symptoms that invalidate H
- $S_t \supset S$, S_t contains the tolerance or special case symptoms that override S_i
- $S_a \supset S$, S_a contains a set of symptoms that enable alternate hypothesis H'
- $S_s \supseteq S$, S_s contains a sufficient set of symptoms necessary to refine H
- $S_{\text{observed}} \supseteq S$, S_{observed} contains all the symptoms observed by the user and presented to the system for activating and refining a hypothesis
- H_r , a hypothesis enabled by the sufficient set to refine H

To refine a particular hypothesis, a flexible match mechanism has been designed. The mechanism allows a degree of match to be specified and modified by the knowledge engineer. Associated with each set of symptoms required to satisfy an enabling,

invalidating, tolerance, refinement or alternate problem condition is a required degree of match. The degree of match is represented as a combined weighting of observed symptoms, S_{observed} , against the symptoms required to satisfy the condition of a set. Thus, to enable or activate hypothesis H , a degree of match for the enabling symptom set, S_e against S_{observed} has to be satisfied. If it is satisfied, the degree of match of S_{observed} against the invalidating symptom set, S_i , is checked to confirm that the observed symptoms do not contradict the hypothesis. If the hypothesis is contradicted to the specified degree, the degree of match of the special case or tolerance symptom set, S_t , is calculated. If the invalidating symptoms are overridden by the tolerance symptoms, H remains enabled and is refined.

To refine H to H_r , the degree of match between the sufficient symptoms for refinement, S_s , and the observed symptoms, S_{observed} , must be satisfied. If the degree of match between S_s and S_{observed} is lower than the required degree, the hypothesis is not refined. Similarly, a degree of match between S_{observed} and the symptoms that enable an alternate hypothesis, S_a has to be satisfied before the alternate hypothesis can be activated. The shared alternate symptoms allow the alternate subgoal to be activated and placed on a queue for subsequent refinement. Thus, the objective of RATIONALE is to continue refining hypotheses of a particular domain until all possible sufficient sets of symptoms are satisfied and a hypothesis can explain the symptoms presented by the user.

3.1 Design

RATIONALE has been designed to operate in three modes. The first mode is a domain definition mode. Here RATIONALE operates as an automated knowledge acquisition tool that allows a knowledge engineer to define the expert system's domain hypotheses hierarchy interactively. The second mode is a domain editing mode where the knowledge engineer can edit the hypotheses definitions. The third mode is the domain run mode where the expert system is run and the end user is given explanations of hypotheses for the observed data.

RATIONALE's top level structure, as illustrated in Figure 3-2, consists of four modules. The main control for the program resides in the finite state machine module. This module controls the states the program goes to as specified by the user's interaction with the interface module, i.e. the window generator module.

Among other things, the state machine module tells the window generator which windows and menus it should pop up. The window generator passes parameters back to it either as flags to indicate user selections, or as frames corresponding to top goal and subgoal edits. RATIONALE's example domain - the ACME workstation advisor - discussed in section 3.4, and illustrated in Figures 3-6 and 3-7, details the formats of the windows, menus and frames in the window generator as well as the window generator's interaction with the state machine module.

The other two modules controlled by the state machine are the frame generator and the explanation generator. The frame generator, described in section 3.3, instantiates the frames that represent the hypotheses and updates them according to user edits. The explanation generator constructs explanations from the reasoning trace and answers questions about the session and the system's capabilities. The explanation generator and its explanation types are discussed in section 3.5.

Figure 3-2: RATIONALE's Top Level Structure

3.2 State Generator

The state machine traverses the sixteen states illustrated in Figure 3-3. State 1 requires users to set a level - *novice*, *experienced* or *expert* - for their interaction with the system and state 2 asks whether they wish to work with an existing domain or to define a new one. If they wish to work with an existing domain, they are asked, in state 3, to select its name from the menu and to indicate, in state 4, whether they wish to edit or run it. If they choose to edit, they are requested in state 5 to indicate whether they wish to edit the top goal or one of the subgoals. The top goal frame is edited in state 6 and subgoals are named and edited in states 7 and 8.

Had the users chosen to define a new domain, they would be asked, in state 9, to specify the name of the new expert system in an edit window, to name, in state 10, the new top level goal, and to edit it in state 6. Once the top level goal is defined, they are asked to name the first top level subgoal and indicate its priority in state 11 relative to the other subgoals. This subgoal is then edited in state 8.

If the users opt to run a system, they are asked, in state 12, to select the top level problem the system should pursue, or to specify *unknown* to indicate that they are not certain about the observed symptom category. If a problem is selected, the possible symptoms required for refining the subgoal are output and the users are prompted for the

Figure 3-3: RATIONALE's Finite State Machine

observed symptoms in state 13. Otherwise, a set of symptoms that allow RATIONALE to enable a particular hypothesis are output and the users are asked to select observed ones. The hypothesis refinement mechanism is described in more detail below. Note that a user who is developing the domain may also edit the goals of the system as results of the session run are produced. Thus, the user may proceed from states 12, 13 or 15 to states 6, 7 or 8 to edit hypotheses. In addition, end users as well as developers of the system may invoke state 1 in order to change the user level from any of states 12 to 15. This allows users to change from a novice to expert level of generated explanation as they require.

State 14, the explanation facility state, can be entered from states 12, 13 and 15. Users are placed in this state whenever they wish an explanation on symptoms or subproblems. Once the system has enough symptoms for refinement from state 13, a list of explanations of the problem based on the observed symptoms can be given.

3.3 Frame Generator

RATIONALE's frame generator module has two main tasks. The first generates the generalized templates for top goals and subgoals. This is done in an object-oriented style as described by Stabler[17]. The facility allows objects and their specific methods, these can either be facts or rules, to be defined. In addition, the facility provides a parent-sibling relationship between the objects. This is very useful for the construction of classification hierarchies for organizing and representing the knowledge in the domain.

The second task in the frame generator instantiates a top goal and respective subgoals for a selected domain definition or edit. The generalized top goal template is shown in Figure 3-4.

```

top goal:
  add_object( ES_name,
              TopG_name,
              [es_name (EN),
              topG_name(TGN),
              es_aim(AIM),
              enabling_condition(problem_type(EC)),
              invalidating_condition(problem_type(DC)),
              tolerance_condition(problem_type(RC))]).

```

Figure 3-4: RATIONALE's Generalized Top Goal

The predicate *add_object (Superclass, Object, [Methods])* adds an object to the tree. *Superclass* is the root, i.e. the expert system, and *Object* is the top goal. *Methods* include the expert system's aim, input as text by the knowledge engineer in the definition stage, and three further conditions that are associated with the object, i.e. an enabling, an invalidating and a tolerance condition. The enabling condition is used to allow diagnosis in one domain over another in a multiple domain expert system. Enabling the top goal allows the refinement of the hypothesis tree or domain hierarchy representing the problem type. The invalidating condition disables the top goal on the basis of an ill-suited problem type for the previously enabled domain. The tolerance condition overrides the invalidating condition for special cases of the problem type. For the example use of the top goal definition in RATIONALE for the ACME workstation advisor domain, refer to Figure 3-7.

The generalized subgoal template, shown in Figure 3-5, is also added to the hypothesis tree as an object with methods. Its Superclass is its parental subgoal which could be either a top level subgoal or a refined subgoal. The subgoal frame includes three explanation slots for the hypothesis it represents. The *novice*, *experienced* and *expert* explanation slots are input by the knowledge engineer when the subgoals are being defined. They may later be edited by a domain expert. An explanation detail slot particular to the hypothesis, holds the expert's numeric estimate of the conceptual complexity of the hypothesis and of its explanatory importance relative to the other hypotheses in the domain hierarchy. This provides a customizable range of detail to distinguish between user levels to output explanations.

The set of all possible symptoms for a hypothesis are grouped to form an enabling, an invalidating and a tolerance subset. These subsets are matched against the user's observed symptoms and their conditions are applied according to the degree of match between the expected symptoms and the ones observed. The enabling condition is expressed in terms of the set or a subset of possible observable symptoms. The invalidating condition is a set of the observed symptoms chosen by the knowledge engineer as invalidating symptoms for the enabled hypothesis. The tolerance condition is a set of special case symptoms that override the invalidating symptoms and allow the hypothesis to be refined further. The subgoal has an alternate-subgoal slot which is triggered by a set of symptoms shared with another hypothesis. The shared symptoms allow the alternate subgoal to be enabled and placed on a queue for subsequent refinement. Once the current hypothesis refinement is complete, the alternate subgoal is selected off the queue and refined as described below. If it can be refined, it is included in the explanation of the observed symptoms.

```

sub goal:
  add_object( SubG_name,
              Refined_subG_name,
              [subG(refined_from,refined_SG),
               subG_name(refined_SG),
               novice_explanation(NE),
               experienced_explanation(EE),
               expert_explanation(EXE),
               explanation_detail(Complexity_of_Subgoal,Importance_Ranking),
               enabling_conditions(Enable_Symptoms,Enabled_degree),
               invalidating_conditions(Invalid_Symptoms,Invalid_degree),
               tolerance_conditions(Special_Case_Symptoms,Tolerance_degree),
               alternate_subG(SubG_name, shared_symptoms(SS),Alternate_degree),
               urgency_indication(Urgency);
              refinement_subG(Refine_to_subG_name, Suff_Symps,Refine_degree)).

```

Figure 3-5: RATIONALE's Generalized Subgoal

An urgency indication is also included in the subgoal frame, thus allowing an urgent problem to be highlighted as such to both the user and RATIONALE. An urgent hypothesis to be refined is placed at the front of the refinement queue. The final slot in the subgoal is the refinement slot. This points to the subgoal to be refined next and to the sufficient symptoms required for its refinement. The refinement mechanism is outlined in section 3.5 and an example of a domain subgoal definition - that of the ACME workstation advisor - is given in Figure 3-6.

3.4 Window Generator

The window generator in RATIONALE is designed to collect information from the user for use by the system. It is a facility that generates windows that display text as well as form-like windows that accept typed information from the user. The generator also has a facility for menus presented to the user for selections using a mouse.

The process by which a developer defines a new expert system in RATIONALE is illustrated in Figure 3-6. RATIONALE first checks the level of the user, so that it may tailor its future explanations to the specified level. For a system that is being edited, RATIONALE could be set to require that the level of the user be an expert with an associated editing password. Such a requirement would ensure that novice and less experienced users would not be permitted to alter the domain knowledge. Should users have comments about the domain knowledge or an object definition, they could input a log or comment to the domain expert and attach it to the object they wish altered.

Once a developer specifies that they wish to work on a new expert system, RATIONALE starts building a new object hierarchy. It first asks the expert for the name of the system.

RATIONALE

Welcome to RATIONALE!
What type of user are you?

User Level
novice
experienced
expert
quit

RATIONALE

What would you like to do? Work on an existing expert system or a new one?

Expert System
existing
new
quit

RATIONALE

What would you like to name this expert system?

name:

Figure 3-6: Defining a New Expert System in RATIONALE

Once the name of the system is established, RATIONALE prompts the user for the details of the top level goal as shown in the example definition of the ACME workstation advisor's top goal in Figure 3-7. The system asks for the name of the goal and displays to the user a form window for the description of the goal. As discussed in the frame generator (see section 3.3), the developer is prompted for: the aim of the system; the conditions that enable the top goal; the conditions that negate the enablement; and the special case conditions that override the negation. In the example, the top level goal specifies that the aim of the expert system is to repair ACME workstation. The enabling condition for the

goal requires that there be a workstation fault. The invalidating condition is the occurrence of a network file server fault whereas the special case tolerance condition is that there be a network fault. Thus, if there is a workstation, a server and a network fault, the top level goal of the ACME workstation expert system is enabled. In a multi-domain expert system running in RATIONALE, there would be several top level goals that are examined one by one to focus on the hierarchy that addresses the user's observed problems.

RATIONALE

What would you like to name the top level goal of FIXIT?

name:

RATIONALE

Please enter details of top level goal **FIX ACME Workstation**?

expert system aim: <u>-fix ACME workstation</u> enabling condition problem type: <u>workstation fault</u> invalidating condition problem type: <u>server fault</u> tolerance condition problem type: <u>network fault</u>	User Wants To save save + quit edit subgoals run expert system quit
--	--

Figure 3-7: Defining a New Top Level Goal in RATIONALE

The top level goal definition is passed to the frame generator from the window generator. The latter instantiates slots in the frames of the objects as they are defined. The subgoal definition proceeds in a similar manner (see Figure 3-8). The user is prompted for the subgoals in their order of importance in the hierarchy, with the left-most nodes being the most important and the right-most nodes the least important (see the sections about subgoal definition - design in section 3.3 and implementation in section 4.2.2).

RATIONALE		
What would you like to name the FIRST priority subgoal of FIXIT?		
name:	SOFTWARE FAULT	quit

RATIONALE							
Please enter details of SOFTWARE FAULT problem of FIXIT ?							
novice explanation: <u>There may be a problem with ACME workstation software run software diagnostics.</u>							
experienced explanation: <u>There may be a S/W corruption, check diagnostics.</u>							
expert explanation: <u>S/W may need a cold restart, test.</u>							
explanation detail:							
relative importance ranking in domain: <u>10</u>							
relative concept complexity ranking: <u>3</u>							
enabling symptoms: <u>[failed boot, memory error, SW process lost]</u>							
required match: <u>0.3</u>							
invalidating symptoms: <u>[hardware error]</u>							
required match: <u>1.0</u>							
special case symptoms to override invalidation:							
<u>[failed boot, memory error, cpu error]</u>							
required match: <u>0.3</u>							
name of alternate subgoal: <u>hardware problem</u>							
shared symptoms with alternate sub goal: <u>[memory error, software process lost]</u>							
required match: <u>0.5</u>							
urgency indication: <u>Priority 1</u>							
name of sub goal to refine to: <u>[disk crash, processor software problem, memory problem]</u>							
sufficient set of symptoms for refinement:							
<u>[[hard disk access denied], [failed boot, memory error], [memory error]]</u>							
required match: <u>[1.0], [0.5], [1.0] 1</u>							
<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th style="padding: 2px 5px;">User Wants To</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px 5px;">save</td> </tr> <tr> <td style="padding: 2px 5px;">save + quit</td> </tr> <tr> <td style="padding: 2px 5px;">edit subgoals</td> </tr> <tr> <td style="padding: 2px 5px;">run expert system</td> </tr> <tr> <td style="padding: 2px 5px;">quit</td> </tr> </tbody> </table>		User Wants To	save	save + quit	edit subgoals	run expert system	quit
User Wants To							
save							
save + quit							
edit subgoals							
run expert system							
quit							

Figure 3-8: Defining a Subgoal in RATIONALE

The priorities can later be edited if the developer is not satisfied with the results. The first form window prompts the user for the name of the subgoal and the second form prompts for the details of the goal.

As detailed in the frame generator, the developer is prompted for the explanations tailored to the user level. The relative importance and complexity of the goal in the hierarchy is also identified. The enabling, invalidating, tolerance, alternate and refinement conditions with their respective required degrees of match are defined. In addition, the urgency indication associated with the occurrence of the problem is assigned. Note that both the top level subgoals and the subsequent refinement subgoals share the same frame definition. Once the subgoal is defined the developer can opt for one of the set of actions in the menu. These actions allow the developer to continue editing goals, running the system or exiting RATIONALE.

Once the object hierarchy of frames is defined, the developer can run the system to test it as well as pass it on to end users and domain experts to further verify its behaviour. To run the system from scratch, the user level is first established as it is in Figure 3-6. If the system developer runs the system once the objects are defined, the user level is automatically set to the one initially specified, expert level in this case. At the start of the run the user is asked to select one of the problems that have been defined as the top level subgoals of the domain (see Figure 3-9). The user is given *unknown* as one of the choices. A set of event-independent or static knowledge explanation options are also displayed to the user in another window entitled **Session Explanation Options**. The explanation options are there to describe to the user the static knowledge and the problem solving strategies of the system.

RATIONALE													
<p>Welcome to FIXIT running in RATIONALE!</p> <p>What type of problem are you having?</p> <p>Please select the name of the problem if you know it, if you don't know it please select unknown.</p>													
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 2px 5px;">FIXIT problems</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px 5px;">software</td> </tr> <tr> <td style="padding: 2px 5px;">hardware</td> </tr> <tr> <td style="padding: 2px 5px;">communication</td> </tr> <tr> <td style="padding: 2px 5px;">power</td> </tr> <tr> <td style="padding: 2px 5px;">unknown</td> </tr> </tbody> </table>	FIXIT problems	software	hardware	communication	power	unknown	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 2px 5px;">User Wants To</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px 5px;">reselect userlevel</td> </tr> <tr> <td style="padding: 2px 5px;">edit sub goals</td> </tr> <tr> <td style="padding: 2px 5px;">edit top goal</td> </tr> <tr> <td style="padding: 2px 5px;">run</td> </tr> <tr> <td style="padding: 2px 5px;">quit</td> </tr> </tbody> </table>	User Wants To	reselect userlevel	edit sub goals	edit top goal	run	quit
FIXIT problems													
software													
hardware													
communication													
power													
unknown													
User Wants To													
reselect userlevel													
edit sub goals													
edit top goal													
run													
quit													

Session Explanation Options
<p>The following are the session explanation questions that RATIONALE can answer. Select the question you wish answered and input the information you are prompted for.</p> <p>ABILITY QUESTIONS</p> <p>How do you deduce this problem?</p> <p>How do you deduce an alternate problem for this problem?</p> <p>How do you focus this problem?</p> <p>FACTUAL QUESTIONS</p> <p>What problems can the system handle?</p> <p>What symptoms and respective degrees of match are</p> <p>a the relevant b /the enabling c /the invalidating</p> <p>d /the special case e /the focusing f /the alternate</p> <p>symptoms and degrees of match for this problem?</p>

Figure 3-9: Selecting an Observed Problem in an expert system run in RATIONALE

RATIONALE																	
<p>RATIONALE is considering RELEVANT SYMPTOMS that help FOCUS your problem. For RATIONALE to achieve this, do one of the following:</p> <p>-> if ALL the RELEVANT symptoms have been observed please select ALL</p> <p>-> if NOT ALL the RELEVANT symptoms have been observed please select only those observed</p>																	
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 2px 5px;">Software Problem SYMPTOMS</th> </tr> </thead> <tbody> <tr><td style="padding: 2px 5px;">hardware_error</td></tr> <tr><td style="padding: 2px 5px;">memory_error</td></tr> <tr><td style="padding: 2px 5px;">cpu_error</td></tr> <tr><td style="padding: 2px 5px;">failed_boot</td></tr> <tr><td style="padding: 2px 5px;">software_process_lost</td></tr> <tr><td style="padding: 2px 5px;">hard_disk_access_denied</td></tr> <tr><td style="padding: 2px 5px;">ALL</td></tr> </tbody> </table>	Software Problem SYMPTOMS	hardware_error	memory_error	cpu_error	failed_boot	software_process_lost	hard_disk_access_denied	ALL	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 2px 5px;">User Wants To</th> </tr> </thead> <tbody> <tr><td style="padding: 2px 5px;">reselect user level</td></tr> <tr><td style="padding: 2px 5px;">reselect problem</td></tr> <tr><td style="padding: 2px 5px;">edit this sub goal</td></tr> <tr><td style="padding: 2px 5px;">edit sub goals</td></tr> <tr><td style="padding: 2px 5px;">edit top goal</td></tr> <tr><td style="padding: 2px 5px;">run</td></tr> <tr><td style="padding: 2px 5px;">quit</td></tr> </tbody> </table>	User Wants To	reselect user level	reselect problem	edit this sub goal	edit sub goals	edit top goal	run	quit
Software Problem SYMPTOMS																	
hardware_error																	
memory_error																	
cpu_error																	
failed_boot																	
software_process_lost																	
hard_disk_access_denied																	
ALL																	
User Wants To																	
reselect user level																	
reselect problem																	
edit this sub goal																	
edit sub goals																	
edit top goal																	
run																	
quit																	
Session Explanation Options																	
<p>The following are the session explanation questions that RATIONALE can answer. Select a question and input the information you are prompted for.</p> <p>EVENT QUESTIONS</p> <p>For my software_problem, WHY do you need these symptoms?</p> <p>For my software_problem, WHY do you NOT need these symptoms?</p> <p>For my software_problem, WHY did you conclude this related problem?</p> <p>For my software_problem, WHY did you NOT conclude this related problem?</p> <p>For my software_problem, HOW did you conclude this related problem?</p> <p>HYPOTHETICAL QUESTIONS</p> <p>What-if these symptoms were also true for my software_problem?</p> <p>What-if this problem replaced my software_problem with these symptoms?</p> <p>ABILITY QUESTIONS</p> <p>How do you deduce this problem?</p> <p>How do you deduce an alternate problem for this problem?</p> <p>How do you focus this problem?</p> <p>FACTUAL QUESTIONS</p> <p>What problems can the system handle?</p> <p>What symptoms and respective degrees of match are</p> <p>a the relevant b /the enabling c /the invalidating</p> <p>d /the special case e /the focusing f /the alternate</p> <p>symptoms and degrees of match for this problem?</p>																	

Figure 3-10: Selecting Relevant Symptoms of Observed Problem

RATIONALE	
<p>Explanations for Problem</p> <p>Your software_problem can be explained as follows:</p> <p>There may be a processor_software_problem which indicates: -->There may be a problem with workstation micro-processor software and it may require re-loading. Run processor diagnostics.</p> <p>There may be a memory_problem which indicates: -->There may be a problem with the memory partition. Try re-loading a new partition to work in.</p> <p>Your software problem has an alternate - a hardware_problem which may be a source of your symptoms.</p> <p>The hardware_problem indicates that: -->There may be a problem with workstation memory card and it may require replacing. Run card diagnostics and replace.</p>	
	<p>User Wants To</p> <p>reselect user level reselect problem edit this sub goal edit sub goals edit top goal quit</p>

Figure 3-11: Example Advice for Observed Problem in RATIONALE

Once a problem is selected off the problem menu, the user is prompted for *relevant* symptoms as shown in Figure 3-10. Again the user is asked for input in the main **RATIONALE** session window. Note that once a problem is established, the explanation options window is expanded to include event-dependent explanations.

After selecting the observed symptoms, **RATIONALE** outputs to the user the explanation and recommended action for the observed problem (Figure 3-11). In addition, the user can select a question from the session explanation window. The mechanisms behind the session reasoning and explanation are described in the sections below.

3.5 Explanation Generator

The explanation generator has two related tasks. The first is to generate hypothesis explanations to advise the user on actions that resolve observed symptoms. The second task is to explain the system's reasoning strategies to the user. This second task is outlined in the section 3.5.2.

3.5.1 RATIONALE's Reasoning Algorithm

The algorithm in Figure 3-12 is used to generate advice or hypothesis explanations. The user is first given a set of top level problems defined as top level sub goals in the domain hierarchy running in RATIONALE. If the user picks a problem to refine, it becomes the current hypothesis and its relevant symptoms are output for selection. Once the user's observed symptoms are selected, the enabling conditions of the observed problem are checked against those required to satisfy the required degree of match. If the problem is enabled from the observed symptoms, the algorithm proceeds. If however, the problem is not enabled, the system proceeds as if the user had selected the problem as *unknown*. If the user selects the problem as *unknown*, the enabling symptom-problem pairs of all the top level problems are output. The user is thus prompted for enabling symptoms relevant to a particular problem.

Figure 3-12: RATIONALE's Reasoning Algorithm

Once a hypothesis is enabled, the problem's invalidating conditions are checked. If the problem is invalidated according to a subset of the observed symptoms, the special case symptoms are then matched against the observed symptoms to check if the tolerance condition can override the invalidating condition. If the tolerance condition succeeds, the problem remains enabled.

If the problem is enabled, the sufficient set of symptoms for refinement are checked. If the refinement conditions are satisfied, the 'problem to refine to' becomes the problem to refine and the process of refinement is repeated. If the hypothesis had no longer been enabled or if the sufficient set of symptoms for refinement had not been satisfied, an alternate problem is pursued. Thus, whether the original problem is refined or not, an alternate problem is checked for. If there is an alternate, it is refined in a manner identical to the originally selected problem. Once refinement is complete, the original and alternate problem explanations are collected. The explanations are collected according to the user level. Thus, if the user is a novice, the novice explanation slot defined in the associated problem frame is collected. Once all explanations associated with the problem, its refinements and its alternates are collected, the explanations are ordered and output to the user as illustrated in Figure 3-11 and the system then waits for the user's next action.

3.5.2 Explaining the Reasoning Strategies

For an expert system to explain its reasoning, it should be built with four objectives in mind. Firstly, be able to tell the user what the system can or cannot do. Secondly, be able to tell the user what the system has done. Thirdly, be able to explain system objectives by telling the user what the system is trying to do. Lastly, be able to tell the user why it is doing what it is in terms of responding to the user's clarification questions.

For RATIONALE to meet these objectives, it has been designed to give the user four types of explanation. The first two types are session sensitive, and they are event and hypothetical explanations. The second two types explain system capabilities, and they are ability and factual questions.

3.5.2.1 Session Sensitive Questions

System event explanations are directly related to the reasoning in the current system session. Event questions that a user is able to ask are directly related to the symptoms and hypotheses the expert system is working with. Event questions can be categorized to *why*, *how* and *what-if* questions.

Event Questions

Why questions are related to the immediate actions the expert system has completed. For example, a *why* question could be: *why the system requires particular symptoms*. *How* questions require explanations of diagnosis methods in relation to the

2

Micro-D

symptoms and hypotheses within the current context. For example, a *how* question could ask: *how the system deduced a particular problem*. The emphasis in these questions are the system's objectives as they are related to its methods. It is also useful for event questions to explain *WHY NOT*. Such questions could ask: *why symptoms were not needed* and *why problems were not deduced*. These give the user an idea of why the system failed to use particular symptoms or deduce particular hypotheses. RATIONALE's event questions are:

For my current problem, WHY do you need these symptoms?
 For my current problem, WHY do you NOT need these symptoms?
 For my current problem, WHY did you conclude this related problem?
 For my current problem, WHY did you NOT conclude this related problem?
 For my current problem, HOW did you conclude this related problem?

RATIONALE's *why* and *how* question menus are shown in Figure 3-10 for an example problem in the ACME workstation advisor. Note that event questions, unlike system capability questions, are displayed once the user has started a session.

Hypothetical Questions

Hypothetical explanations are often referred to as *what-if* explanations. They explain to the user the result of adding new symptoms to an established problem. Such a query could ask: *what-if new symptoms were added to the established problem*. They also explain to the user the result of replacing an established problem with a new one with a modified set of symptoms. For example, the question could ask: *what-if a new problem with new observed symptoms replaced the established one*. Hypothetical explanations can be both event dependent and independent. In the event dependent case, new symptoms are

associated with an already established problem. In the event independent case, a new problem with new symptoms could replace the established problem. In RATIONALE, hypothetical explanations are generated using a second history of the session in which the user defines hypothetical symptoms and hypotheses. The difficulty in providing such a facility is the ability to return to the original user session without confusing the two sets of symptoms and hypotheses. Such a facility is valuable for users testing the knowledge of the system. Hypothetical explanations, unlike Event explanations, are not readily available in most expert systems. This is mainly due to the overhead associated with tracking a user's current and hypothetical sessions. RATIONALE's hypothetical questions are:

What-if these symptoms were also true for my current problem?
 What-if this problem replaced my current problem with these symptoms?

RATIONALE's what-if type questions are shown in Figure 3-10 for an example problem in the ACME workstation advisor.

3.5.2.2 System Capability Questions

Questions about system capabilities allow the user to get explanations about system methodologies independent of any particular context. A user can question the system about the use of symptoms and problems. These include questions such as: *how does the system use specified symptoms and how are certain problems deduced.*

Ability Questions

Ability questions are independent of user sessions and can be asked at any time, thus giving the user an insight into what the system is capable of. This is why in RATIONALE capability questions can be asked before a session is started. These explanations are useful in aiding the user to test as well as learn the methodologies of the system. Ability questions allow the user to ask questions about what the system can do with particular symptoms to arrive at particular hypotheses. The questions that RATIONALE explains are:

How do you deduce this problem?
How do you deduce an alternate problem for this problem?
How do you focus this problem?

RATIONALE's ability question menu is shown in Figure 3-9.

Factual Questions

Factual questions are similar to ability questions. They differ in that they allow the user to query the system's static knowledge directly. Such questions include: *what problems can the system handle* and *what can a problem be refined to*. This permits a user to ask about the system's capability in using certain symptoms and problems to arrive at certain results. An example of such a query is: *what are the conditions to satisfy a particular problem*.

RATIONALE's factual questions are:

What problems can the system handle?

What symptoms and respective degrees of match are

a) the relevant b) /the enabling c) /the invalidating

d) /the special case e) /the focusing f) /the alternate

symptoms and degrees of match for this problem?

RATIONALE's factual question menu is illustrated in Figure 3-9.

3.5.2.3 Generated Explanations

Generated explanations in current expert systems are typically clumsy utterances and ignore most computational models of discourse. Ideally, generated explanations should be considered as important as providing the explanation capability itself. There are six important considerations for generated explanations that facilities would benefit from adhering to. Illustrations of these six considerations are given below demonstrating the power of RATIONALE's explanations. RATIONALE's explanations are template-based. This form of explanation connects pieces of text to variables that are instantiated from the knowledge in the system. This allows explanation templates to be independent of the domain running in RATIONALE. It also simplifies the task of generating dynamic explanations according to the current context. The precise format of the templates is elaborated in the implementation section of the thesis. Several examples that demonstrate the use of the explanation templates for an example domain defined in RATIONALE (the ACME workstation advisor) have been included in Appendix B.

Contextually Dependent References

Once a context is introduced in a user question, the generated explanation should reference it much as a human explaining within a particular context would. An example of the explanation templates in RATIONALE for a session dependent question follows. (Note that the / separates the template phrase choices - the choices are made based on the current context of the session.)

User Question: Why did you not deduce *subproblem B*?

Explanation: *Subproblem B* could not be deduced since /

the alternate subproblem symptoms *S*, of current *subproblem A* could not be concluded /

the sufficient focusing symptoms *SS* of current *subproblem A* could not be concluded /

none of the selected symptoms could lead to it.

The use of the template can be illustrated based on the domain subgoal defined in RATIONALE for the ACME workstation advisor in Figure 3-8:

User Question 1: Why did you not deduce a *hardware problem*?

Explanation: A *hardware problem* could not be deduced since the alternate subproblem symptom *memory error*, of the current *software problem* could not be concluded.

User Question 2: Why did you not deduce a *processor software problem*?

Explanation: A *processor software problem* could not be deduced since the sufficient focusing symptoms *failed boot* and *memory error* of the current *software problem* could not be concluded.

Anaphoric References

These are references to an already introduced entity. Thus, users could ask questions with anaphoric references and the explanation facility would be expected to understand the reference and hence deduce the entity referred to. Continuing with the introduced example:

User Question: How did you use selected symptoms?

Explanation: Selected symptoms *S* were used as /
enabling symptoms for *subproblem A* /
tolerance symptoms for *invalidating*
 symptoms *Si* of *subproblem A* /
alternate symptoms for *alternate*
subproblem B of *subproblem A* /
focusing symptoms for *refining*
subproblem A to *subproblem C*.

Continuing with the domain subgoal defined in Figure 3-8:

User Question: How did you use selected symptoms?

Explanation: Selected symptoms *failed boot* and *software process lost* were used as:
 ---> *enabling* symptoms for the *software* problem,
 ---> *alternate focusing* symptoms for the *alternate*
 problem, *hardware* problem,
 ---> and *focusing* symptoms for *refining* the
 software problem to a *processor software*
 problem.

Elliptic References

Such references involve indirect references to introduced entities. Again such interpretation of user questions would follow the human model of explanation. An example of an elliptic reference is:

User Question 1: Why did you not deduce *subproblem B*?
User Question 2: What were its symptoms?
Explanation: *Subproblem B's* enabling symptoms are S_e .

Echo User's Question

This is the ability to include several messages in one generated explanation. Such an explanation could echo the user's question subtly to assure the user that their intentions were understood. This is illustrated in the explanations generated above.

Clarification Questions

Questions about the user's intent should be specific and well directed and understandable by the facility itself. Clarification is mostly applicable to a dialog-based interface versus a menu or forms-based interface such as RATIONALE's. An example of clarification is:

User Question: What is the basis of *hypothesis A* ?

Clarification: Do you mean, _
 what are the symptoms for *hypothesis A* ?

 Versus

 Clarify '*basis*' please?

In the latter, the system's request for clarification is open ended which may leave the user confused as to the system's expectations in terms of a response. It is imperative for the facility that requires explanation clarification questions to be able to question the user in a meaningful manner. This implies that for the user's response to be meaningful to the explanation facility itself, the explanation facility's question formulation should be based on what it itself can understand.

Adhering to these considerations for generating explanations may seem ambitious for some systems. But such context sensitive help and knowledge debugging aids in explanation facilities are fast becoming a requirement. The author believes that users will soon reject facilities that generate clumsy and cryptic explanations in any knowledge-based application.

Chapter 4

Implementation of RATIONALE

To demonstrate the most important concepts of the explanation mechanisms and their underlying structure, RATIONALE's run mode has been implemented. The definition and edit modes of the tool are to be implemented in a later phase of RATIONALE.

The run mode of the RATIONALE tool has three main components. The first is the frame generator module which builds the object hierarchy and provides the top goal and subgoal object definitions. The second is the hypothesis explanation generator which collects the hypotheses explanations corresponding to the user's observed symptoms. The third component is the session explanation component which provides the user with event, hypothetical, ability and factual explanations.

In addition to the three tool components, RATIONALE has been implemented to run an example domain. The example domain is the diagnosis of problems with a generic workstation that is modeled after the LISP workstations that are currently used to develop Artificial Intelligence applications. The sections below describe the implementation of the run mode of RATIONALE.

On the Use of Quintus Prolog

Quintus Prolog running on a Xerox 1109 is the first environment that RATIONALE runs in. RATIONALE also runs in Quintus on a Sun 3/60. The current interface of the system is not fully window and menu-based. As the examples in Appendix B illustrate, the menus are generated in a textual fashion and the user makes a selection either by typing a choice directly or by dragging the mouse across the selection and copying it into the prompt. Menus and windows are not part of the first implementation due to their unavailability in Quintus Prolog. However, the program is written in a modular fashion which groups together user options so that the options can be taken directly to a menu-based interface when one is available. In addition, the modules group together the text for the prompts so that collection of prompts can be written to a window along with the selection menus.

Another consideration in the Quintus Prolog implementation is the readability of the code. The following programming styles have been adopted:

first, to better illustrate the value of a variable, a variable is sometimes read in and then checked rather than simply bringing its value in without identifying its name,

`a(X) :- X=1, ...` Versus `a(1) :- ...`

second, to check the length of a list, the predicate `length` is used,

`a(X) :- length(X,1)` Versus `a([Y]) :- ...`

third, case statements are sometimes used in the code to better illustrate if-then-else conditions, where the symbol $\rightarrow$ implies *then* and the symbol $|$ implies *else*,

```
a(X) :-  X=1 -> ...
        | X=2 -> ...
        | X=3 -> ...
        | otherwise ...
```

Versus

```
a(1) :- ...
a(2) :- ...
a(3) :- ...
```

4.1 Example Domain

The domain chosen to illustrate the operation of RATIONALE has the top level goal of *fixing ACME workstation*. This goal can be refined to determine the source of the ACME workstation problem. The sources of the problem are represented as hypotheses concepts in the diagnosis hierarchy of the domain. They are prioritized according to their importance to the typical diagnosis. The leafs of the domain hierarchy are prioritized from left to right with the leftmost node having the highest priority. The prioritized order of diagnosis for the ACME workstation is a *software problem*, a *hardware problem*, a *communication problem* and lastly, a *power problem*. Each problem can be refined further as illustrated in the ACME domain hierarchy of Figure 4-1.

The *software problem* can be refined to a *disk crash*, a *microprocessor software problem* or a *memory problem*. Similarly, a *hardware problem* can be refined to a *processor card problem* or a *memory card problem*. The top level subgoal refinements can themselves be refined further. For example, a *disk crash* can be refined to a *disk sector problem*.

The concept of alternate subgoal is an important concept in the definition of the domain. The coordinated numerals in Figure 4-1 are intended to illustrate those subgoals that are alternates. A *software problem* is an alternate top level subgoal to a *hardware problem*. Thus, if a user has symptoms indicating a *software problem*, the *hardware problem* is investigated as a possible alternate. Alternates can exist at each level between any leafs in the domain hierarchy. For example, a *disk crash*, a *microprocessor software problem* and a *processor (CPU) card problem* are all alternates although they do not all share the same immediate parent.

Figure 4-1: ACME Workstation Hypothesis Hierarchy

4.2 Frame Generator

Two types of frames are generated in RATIONALE. They are the top goal and the subgoal frames. As illustrated in the design section 3.3, the frames are defined in a generic manner and are instantiated by concepts defined in the domain hierarchy of a particular application.

4.2.1 Top Goal Frame

The top goal is defined by this rule implemented in Quintus Prolog:

```
add_topgoal(ES_name,
            TG_name,
            AIM,
            problem_type(EC,DC,RC))
:-
    create_root,
    add_object(expert_sys(ES_name),
               subG_name(TG_name),
               [top_goal(ES_name,TG_name),
                tg_name(TG_name),
                es_aim(AIM),
                enable_cond(problem_type(EC,_)),
                invalid_cond(problem_type(_,IC_)),
                tolerance_cond(problem_type(_,TC))]).
```

The first clause of the rule simply creates the root of the hierarchy which acts as a marker for the top level goal of the expert system. The *add_object* predicate holds the (parent, sibling, methods) information for the top goal object. The parent information in this case is the expert system that the top goal is being defined for. The sibling is the name

of the top level goal. The first two methods provide the parent-sibling relation while the third method provides the explicit aim of the expert system. The aim is a string of text that the developer of the expert system writes to explain the main objective of the system. The next three methods are best illustrated with an example definition. Consider the definition for ACME's top level goal:

```
add_topgoal(acme,
            fix_workstation,
            'fix acme workstation',
            problem_type(workstation,server,network)).
```

The three methods of the top goal are the enabling, invalidating and tolerance conditions. The values for their respective predicate variables are derived from the definition of the *problem_type* predicate. The enabling condition predicate holds the variable *EC* which is defined as the triggering condition for the expert system. For ACME, the variable *EC* would be instantiated to the atom *workstation*. *EC* can be a list or an atom depending on the number of conditions one wishes to define. The enabling condition provides one with an entry point to the expert system. If this condition is not true or is invalidated by the subsequent conditions, then the expert system hierarchy below the top level goal is not accessed for a diagnosis of the user's problem. In a distributed expert system with several domain hierarchies, the enabling condition is used to focus in on the domain appropriate for the observed symptoms.

The invalidating condition is determined by the value of the variable *IC*. *IC* defines negating evidence and is instantiated to *server* in the ACME domain instance. Hence, if the user observes the invalidating condition, *server problem*, the expert system would not be enabled on the basis of the observed symptom of a *workstation problem*.

The tolerance condition provides the special case condition or tolerance condition, *TC*, to be applied against the invalidating condition. In ACME this is defined as *network*. If the top goal is enabled and subsequently invalidated, then the tolerance condition *network problem* must be observed by the user for the special case condition to be satisfied and for the top level goal to be refined further.

4.2.2 Subgoal Frame

The second type of frame used in RATIONALE is the subgoal frame. The subgoal frame, like the top goal frame, is defined as a Prolog rule and has the following format:

```
add_subgoal(SG_name_refined_from,
            Ref_SG_name,
            explanation(NoviceE,ExperiencedE,ExpertE),
            explanation_detail(SubG_Concept_Complexity,Detail_Importance),
            enabling_cond(Enabling_Symps,EDegree),
            invalidate_cond(Invalid_Symps,IDegree),
            tolerate_cond(Special_Case_Symps,TDegree),
            alternate_subG(Alternate_SG_name,Shared_Symps,ADegree),
            Urgency,
            refinement_subG(Refine_to_SG_name,Suff_Set,RDegree))
:-
add_object(subG_name(SG_name_refined_from),
            refined_SG(Ref_SG_name),
            [sub_goal(SG_name_refined_from,Ref_SG_name),
            ref_subG_name(Ref_SG_name),
            novice_explanation(NoviceE),
            experienced_explanation(ExperiencedE),
            expert_explanation(ExpertE),
            explanation_detail(SubG_Complexity,Detail_Importance),
            enabling_cond(Enabling_Symps,EDegree),
            invalidate_cond(Invalid_Symps,IDegree),
            tolerate_cond(Special_Case_Symps,TDegree),
            alternate_subG(Alternate_SG_name,Shared_Symps,ADegree),
            urgency_indication(Urgency),
            refinement_subG(Refine_to_SG_name,Suff_Set,RDegree)]).
```

The *add_object* predicate is used to define the subgoal as a (parent,sibling,methods) triplet. The parent is the parent object in the domain hierarchy and the sibling is the object being defined. Consider the subgoal definition for the *software* object below:

```
add_subgoal(fix_workstation,
            software_problem,
            explanation('There may be a problem with ACME workstation software, run the
                        diagnostics',
                        'Workstation software problem, check diagnostics',
                        'Workstation sw bug or corruption, run tests'),
            explanation_detail(3,10),
            enabling_cond([failed_boot,memory_error,sw_process_lost],0.3),
            invalidate_cond([hardware_error],1.0),
            tolerate_cond([failed_boot,cpu_error,memory_error],0.3),
            alternate_subG(hardware_problem,[memory_error,sw_process_lost],0.5),
            3,
            refinement_subG([disk_crash,processor_sw_problem,memory_problem]
                           [[hard_disk_access_denied],[failed_boot,memory_error],[memory_error]],
                           [1.0,0.5,1.0])).
```

The parent is the top goal *fix_workstation* and the sibling is the object *software_problem*. Each subgoal object defined in the domain hierarchy has three levels of explanations represented as methods. These explanations provide an explicit reason for the existence of the object in the diagnostic hierarchy. Three levels of explanations are derived from the three methods, they are: *novice*, *experienced* and *expert* explanations. The *novice* explanation is more detailed than the other two levels since novice users usually require more guidance [Hayes83]. Encoding the explanations of the objects directly in the frames provides RATIONALE with a convenient source of recommendations to be made to the user at the end of a diagnostic session. The explanations are strategically captured by RATIONALE when the diagnostic hierarchy of the domain is being defined and the reasoning strategies are being determined by the developer based on the domain expert's view.

The *explanation_detail* method captures a range of explanation that is associated with each object. Its two arguments *subgoal complexity* and *detail importance* provide RATIONALE with a method to tailour explanations to the three selected levels. If a subgoal is being explained to a novice user and it has a higher complexity than importance ranking, then the explanation is simplified by omitting detail such as the less important nodes traversed to conclude the diagnosis and the user is given the least complex but important detail. The algorithm that controls this is described in section 4.3.

Similar to the top goal methods, the subgoal has enabling, invalidating and tolerance conditions. Consider the example definition of the *software_problem* object. The enabling condition requires one of three symptoms to be present. The invalidating condition requires the symptom *hardware_error* to be present and the tolerance condition requires any of three other symptoms to be present. Thus, if the software problem is enabled and invalidated, the tolerance symptoms are checked for, and the subgoal is refined if the tolerance conditions are satisfied. If the tolerance conditions are not satisfied, the subgoal is not refined.

The *refinement_subG* method provides the condition for refinement. In the software problem object, the *refine-to-subgoal* is a list of three possible subgoals to refine to. Each of these has an associated list of sufficient symptoms for refinement and a required degree. Thus, if the software problem is refinable, a *disk_crash* subgoal would be the first to be examined for refinement.

The subgoal definition of the disk crash problem is the following:

```
add_subgoal(software_problem,
  disk_crash,
  explanation('There is a problem with workstation hard disk software and it may require
    reloading. Check hard disk diagnostics.',
    'Hard disk software problem, run disk diagnostics.',
    'Hard disk sw fault needs tracing'),
  explanation_detail(5,4),
  enabling_cond([hard_disk_access_denied,sw_process_lost],0.5),
  invalidate_cond([power_error],1.0),
  tolerate_cond([connector_problem],1.0),
  alternate_subG(cpu_card_problem,[keyboard_locked,screen_blank,mouse_failing],1.0),
  3,
  refinement_subG(disk_sector_problem,[hard_disk_access_denied],1.0)).
```

If the disk crash subgoal itself is found refinable, it is refined to a *disk_sector_problem*.

The disk sector problem is defined below and has no refinements:

```
add_subgoal(disk_crash,
  disk_sector_problem
  explanation('There is a problem a sector of the hard disk of the workstation and it may
    require a hard disk overhaul. Run hard disk sector diagnostics and contact
    ACME service department.',
    'Hard disk sector problem, run disk diagnostics and call for service.',
    'Disk sector fault needs tracing, call service with results.'),
  explanation_detail(8,7),
  enabling_cond([hard_disk_access_denied,sw_process_lost],0.5),
  invalidate_cond([power_error],1.0),
  tolerate_cond([connector_problem],1.0),
  alternate_subG(none,[none],0.0),
  3,
  refinement_subG(no_refine_to_subG,[no_suff_sct],0.0)).
```

Another method represented in the subgoal object is the *Urgency* indication. This indication is used to re-order the user recommendations once a diagnosis is determined. In the case of the software problem, if all three refinement subgoals are found valid, the

recommendations made by RATIONALE are then ordered according to the *Urgency* indication.

The alternate subgoal method is similar to the refinement subgoal method. It has a set of shared symptoms that are compared to the observed symptoms. In the *software problem* object, the alternate problem is a *hardware problem* and it has an associated set of symptoms that are required to validate it as an alternate. The alternate problem provides another set of possible diagnoses based on observed symptoms and the alternate is refined as far as the domain hierarchy allows it to be refined.

RATIONALE's frame generator provides a flexible utility for defining objects in a diagnostic domain hierarchy. To run any application in RATIONALE, one only needs to create the object hierarchy specific to the application. The object definitions form the knowledge base of the domain running in RATIONALE. They group together knowledge about the concepts in the domain and are very easy to modify. Once the hierarchy is instantiated as a set of goal frames, RATIONALE can be used as shell to run the application. Thus providing the diagnoses and explanations specific to the application. A full definition of the ACME hierarchy is provided in Appendix A for the reader's convenience.

4.3 Explanations

RATIONALE's explanations are generated by three separate modules. The first of these is the main module which controls the traversal between states (see section 4.3.1). The second module is the reasoning algorithm module which provides recommendations to the user based on observed symptoms and corresponding hypotheses (see section 4.4). The third module is the explanation of reasoning strategies module which answers user questions about session recommendations (see section 4.5).

4.3.1 State Traversal

The main module starts up the operation of RATIONALE according to the following predicate:

```

rationale(Domain):-
    start(Domain),
    select_userlevel(Userlevel),
    select_subproblem(Domain,SUBP,Enable_mode),
    select_symptoms(Domain,SUBP,SYMPTOMS),
    explainer(Userlevel,Domain,SUBP,SYMPTOMS,Enable_mode,
              Refinements,Sibling_Refinement,Alternate,AlternateR),
    get_next_action(Userlevel,Domain,SUBP,SYMPTOMS,
                    Refinements,Sibling_Refinement,
                    Alternate,AlternateR,Enable_mode).

```

The first clause welcomes the user to RATIONALE running a selected Domain and the second clause places the user in the first state. The first state is intended to determine the user level as one of [novice,experienced,expert] depending on their familiarity with the

Domain. Note that for the reader's convenience, several example test runs that demonstrate the operation of RATIONALE running the ACME domain have been included in Appendix B.

State1 - Select the User Level

The user level selection is coded as:

```
/* state 1 */
/* -----user level----- */

select_userlevel(Userlevel) :-
    prompt(['
        What type of user are you ?
        novice
        experienced
        expert']),
    prompt(['
        Please type in novice if you are a new user, experienced if you are an
        occasional user and expert if you are very familiar with the system.']),
    prompted_line(['
        user level: ',Userlev_code),
    name(Userlevel, Userlev_code),
    memberchk(Userlevel,[novice,experienced,expert]),
    prompt(['Your user level has been set to ',Userlevel,'.']).
```

The same user level predicate is re-visited whenever the user wishes to re-select it. For example, the user level can be re-selected in the case of a user who had selected a *novice* level opting for *experienced* or *expert* explanations if the resultant explanations are too detailed.

State 12 - Select a User Problem

- Once the user level is set, selecting a subproblem is the next state that the main program proceeds to. This is state 12 previously described in the design of the state generator section 3.2. The names of the top level hypotheses of the Domain are output to the user for a choice. Once the user types in a choice, it is matched against the names of all the hypotheses in the domain. The reason that both the top level subgoal and the refinement goal names are checked is to allow more experienced users to focus their problem more rapidly. Thus, if an expert knows that the problem in ACME is a *disk crash*, RATIONALE starts focusing at the chosen node rather than its parent node, the *software problem*.

```

/* state 12 */
/* -----select subproblem ----- */
select_subproblem(Domain,SUBP,Enable_mode):-
  what_subGs_exist(Domain,TopGNames),
  prompt(['What type of problem have you been having with ', Domain,'?']),
  writeList(TopGNames),
  prompt(['Please type in the name of the problem if you know it. If you do
          not know it, type in unknown.']),
  prompted_line(['type of problem: ',ObservedSUBP_code),
  name(ObservedSUBP, ObservedSUBP_code),
  what_subPs_exist(AllNames),
  know_subproblem(ObservedSUBP,AllNames,Names,SUBP,Enable_mode).

```

A user could also choose *unknown* as their problem selection. If this is selected, RATIONALE outputs to the user *problem-enabling symptom* pairs to select from. Thus, if they are unsure of what the problem may be, they are assisted by a list of problem enabling symptoms which are used to trigger the refinement at a top level branch in the hierarchy.

The *know_subproblem* predicate below checks if the user selected a problem from the Domain hierarchy. If the problem is *unknown*, it proceeds to enable one.

```

/* -----check user's subproblem selection ----- */

know_subproblem(ObservedSUBP,AllNames,Names,SUBP,Enable_mode):-
  memberchk(ObservedSUBP,AllNames),
  (ObservedSUBP = SUBP),
  (Enable_mode = user_selected). /* if SUBP is known go to state 13 */

know_subproblem(ObservedSUBP,AllNames,Names,SUBP,Enable_mode):-
  \+memberchk(ObservedSUBP,AllNames),
  enable_subproblem(Names,SUBP,_), /* SUBP unknown,enable one */
  (Enable_mode = system_deduced).

```

To enable a subproblem, the predicate *what_enabling_conds_exist* examines the enabling conditions of the hypotheses in the domain hierarchy. It does this by extracting the enabling condition slot information from each top level object's frame. Once the enabling conditions are collected, the user is prompted for the name of the subproblem and its respective enabling conditions. After these are input, the observed symptoms are checked against the enabling ones and the selected problem is triggered if the degree of match is satisfied.

If the match is not satisfied, the user is prompted for the *problem-symptom* pairs again.

```

/* -----enable a problem----- */

enable_subproblem(Names,SUBP,Enable_Symps):-
  what_enabling_conds_exist(Names,[ECondlEConds]),
  prompt(['RATIONALE is considering PROBLEM-SYMPOMS pairs to
    help determine your type of problem. For RATIONALE to focus
    on the type of problem you have, do one of the following:']),
  prompt(['If the symptoms listed below have all been observed, please type in
    the problem name, carriage return, and all.']),
  prompt(['If not all the listed symptoms have been observed, please type in
    the problem name, carriage return, and only those symptoms
    observed separated by blanks and ending with a period.']),
  pretty_print(Names,[ECondlEConds]),
  prompted_line(['name of problem: ',SG_code),
  name(SG,SG_code),
  prompt(['Please input the symptom names separated by blanks and ending
    with a period.']),
  prompt(['symptoms of ',SG,: '']),
  read_in(SYMS),
  delete(SYMS,,Observed_Symps),
  (memberchk(all,Observed_Symps)
  -> SG = SUBP,
    enabling_confd(refined_SG(SG),ECs,_),
    Enable_Symps = ECs
  | enable_check(SG,Observed_Symps)
  -> SG = SUBP,
    Enable_Symps = Observed_Symps
  | otherwise
  -> prompt(['Sorry, a ',SG,' cannot be deduced from the restricted set of
    symptoms you gave.']),
    enable_subproblem(Names,SUBP,Enable_Symps)
  ).

```

State 12, similar to state 1 can be re-visited if the user wishes to re-select a problem. Once a problem is selected, the transactions of state 12 are complete and control is passed on to state 13, the selection of relevant symptoms.

State 13 - Select Symptoms Relevant to Problem

The set of relevant symptoms is formed by the predicate *combined_set*. This predicate forms a set of all enabling, invalidating, tolerance, refinement, and alternate problem symptoms. The purpose of the predicate is to output to the user a list of symptoms related to the refinement of a particular hypothesis. The symptoms that satisfy the various conditions are extracted from the selected problem's frame slots.

Once the combination of relevant symptoms for the selected problem is found, the user is prompted to select a subset of the relevant symptoms. The user could select *all* as the observed symptoms and all the relevant symptoms are selected for the hypothesis. If none of the symptoms are observed or if the observed symptoms are not relevant to the selected problem, the system returns to state 12 and the user is prompted for a new *problem* or a new selection of *problem-symptom* pairs.

```

/* state 13 */
/* ----- select relevant hypothesis symptoms ----- */

select_symptoms(Domain,SUBP,SYMPTOMS):-
    combined_set(SUBP,SYMPS),
    prompt(['RATIONALE is considering RELEVANT SYMPTOMS that help
           FOCUS your problem further. For RATIONALE to achieve this,
           do one of the following:']),
    prompt(['If the RELEVANT symptoms listed below have all been observed,
           please type all.']),
    prompt(['If not all the RELEVANT listed symptoms have been observed,
           please type in only those symptoms observed separated by blanks
           and ending with a period.']),
    write('RELEVANT SYMPTOMS of '),
    write(SUBP),
    writeList(SYMPS),
    prompt(['Please input the symptom names separated by blanks and ending
           with a period.']),
    prompt(['OBSERVED symptoms of ',SUBP,': ']),
    read_in(SYMS),
    delete(SYMS,Observed_Syms),
    (memberchk(all,Observed_Syms) -> (SYMPS = SYMPTOMS))
    | subsets(Observed_Syms,SYMPS) -> (Observed_Syms = SYMPTOMS)
    | otherwise -> prompt(['Your ',SUBP,' cannot be focused further from the
           restricted set of symptoms you have observed.']))

```

```

/* -----combined subproblem symptoms ----- */
/* This predicate forms a unique set of symptoms associated with a
particular hypothesis already instantiated in the hierarchy. Its form is:
combined_set(SubgoalName,All_SympS).

```

```

SubgoalName-----> Hypothesis in tree whose symptoms are required.
All_SympS-----> Set of all symptoms associated with hypothesis
*/

```

```

combined_set(SubgoalName,All_SympS):-
  enabling_cond(refined_SG(SubgoalName),Set1_),
  invalidate_cond(refined_SG(SubgoalName),Set2_),
  tolerate_cond(refined_SG(SubgoalName),Set3_),
  refinement_subG(refined_SG(SubgoalName),RefTo,RefSympS_),
  alternate_subG(refined_SG(SubgoalName),Alt,AltSympS_),!,
  combine_subs(Alt,AltSympS,Set4),
  combine_subs(RefTo,RefSympS,Set5),
  union(Set1,Set2,Set1_2),
  union(Set1_2,Set3,Set1_3),
  union(Set1_3,Set4,Set1_4),
  union(Set1_4,Set5,AllS),
  check_Nos(AllS,All_SympS).

```

```

combine_subs(SG,SGSympS,S):-
  atom(SG),
  S = SGSympS.

```

```

combine_subs(_,SGSympS,S):-
  flatten(SGSympS,SGS),
  make_set(SGS,S).

```

```

make_set(SGS,S):-
  is_set(SGS)
  -> S = SGS
  | list_to_set(SGS,S).

```

```

check_Nos(AllS,All_SympS):-
  check_no(AllS,IntermedS),
  check_no_suffS(IntermedS,All_SympS).

```

```

check_no(AllS,IntermedS):-          /* check if slot has 'none' as symptoms */
  member(none,AllS)
  -> del_element(none,AllS,IntermedS)
  | IntermedS = AllS.

```

```

check_no_suffS(IntermedS,All_SympS):- /* check if slot has 'no_suff' S */
  member(no_suff_set,IntermedS)
  -> del_element(no_suff_set,IntermedS,All_SympS)
  | All_SympS = IntermedS.

```

State 15 - Refine Selected Problem

State 15, the refinement of the selected problem according to the observed relevant symptoms follows State 13. The refinement state is contained in two sub-modules, the first is *refine_subproblem* and the second is *generate_explanations*. The two modules are called from the predicate *explainer* by the main control module. State 15 is described in section 4.4, this is the *explainer* predicate that triggers problem refinement:

```
explainer(Userlevel,Domain,SUBP,SYMPTOMS,Enable_mode,
          Refinements,Sibling_Refinement,Alternate,AlternateR):-
  refine_subproblem(Domain,SUBP,SYMPTOMS,Enable_mode,
                    Refinements,Sibling_Refinement,Alternate,AlternateR),
  generate_explanations(Userlevel,SUBP,Refinements,Sibling_Refinement,
                        Alternate,AlternateR).
```

Get Next User Action: 1 of 5

After state 1, 12, 13 and 15 are visited by the main control module and the user is given a diagnosis based on the observed hypotheses symptoms, the user is given a choice of five actions. The choice of which action and result is controlled by the predicate *get_next_action*:

```
get_next_action(Userlevel,Domain,SUBP,SYMPTOMS,
                Refinements,Sibling_Refinement,
                Alternate,AlternateR,Enable_mode):-
  nl,prompt(['What would you like to do next? ']),
  nl,prompt(['1. re-select user level and re-display results ']),
  nl,prompt(['2. re-select problem ']),
  nl,prompt(['3. re-select symptoms for ',SUBP]),
  nl,prompt(['4. ask questions about session results ']),
  nl,prompt(['5. quit RATIONALE running ',Domain]),
  nl,prompted_line(['user wants to: '],Nextcode),
  name(Next,Nextcode),
  get_action(Next,Userlevel,Domain,SUBP,SYMPTOMS,
            Refinements,Sibling_Refinement,
            Alternate,AlternateR,Enable_mode).
```

The decision of which action the user wishes is controlled by *get_action*. It simply sends the system to the state the user wishes to proceed to. Examples of the use of the main menu options of RATIONALE are included in Appendix B1.

1 *Re-Select User Level*

The first state is the re-selection of the *user level*. Here, the user is given the option to re-select the user level and re-display the diagnoses of the observed hypotheses symptoms. If action 1 is chosen, the user level is re-selected in state 1 and the *explainer* predicate which traverses to state 15 is called.

```
get_action(Next,Userlevel,Domain,SUBP,SYMPTOMS,
           Refinements,Sibling_Refinement,
           Alternate,AlternateR,Enable_mode):-
    Next = 1,
    reselect_userlevel(Newlevel),
    explainer(Newlevel,Domain,SUBP,SYMPTOMS,Enable_mode),
    get_next_action(Userlevel,Domain,SUBP,SYMPTOMS,
                   Refinements,Sibling_Refinement,
                   Alternate,AlternateR,Enable_mode).
```

2 *Re-Select Problem*

The second action that the user could request is the re-selection of the problem. The user is placed in state 12 to make a new problem selection and the system then proceeds to state 13 to select new symptoms and to state 15 to refine the re-selected problem.

```
get_action(Next,Userlevel,Domain,SUBP,SYMPTOMS,
           Refinements,Sibling_Refinement,
           Alternate,AlternateR,Enable_mode):-
    Next = 2,
    select_subproblem(Domain,NewSUBP,Enable_mode),
    select_symptoms(Domain,SUBP,NewSYMPTOMS),
    explainer(Userlevel,Domain,NewSUBP,SYMPTOMS,Enable_mode),
    get_next_action(Userlevel,Domain,SUBP,SYMPTOMS,
                   Refinements,Sibling_Refinement,
                   Alternate,AlternateR,Enable_mode).
```


3 Re-Select Symptoms

The user can also re-select symptoms for the same problem. Here the symptoms are re-selected using *select_symptoms* in state 13 and the problem is refined using *explainer* in state 15.

```
get_action(Next,Userlevel,Domain,SUBP,SYMPTOMS,
           Refinements,Sibling_Refinement,
           Alternate,AlternateR,Enable_mode):-
    Next = 3,
    select_symptoms(Domain,SUBP,NewSYMPTOMS),
    explainer(Userlevel,Domain,SUBP,NewSYMPTOMS,Enable_mode),
    get_next_action(Userlevel,Domain,SUBP,SYMPTOMS,
                   Refinements,Sibling_Refinement,
                   Alternate,AlternateR,Enable_mode).
```

4 Explain Reasoning Strategies

One of the main actions the user can select is to receive explanations on the results and reasoning strategies used in the session. This action triggers state 14, the session explanation module, described in detail in section 3.2.3.3 below.

```
get_action(Next,Userlevel,Domain,SUBP,SYMPTOMS,
           Refinements,Sibling_Refinement,Alternate,AlternateR,
           Enable_mode):-
    Next = 4,
    sessionE(Userlevel,Domain,SUBP,SYMPTOMS,
             Refinements,Sibling_Refinement,
             Alternate,AlternateR,Enable_mode).
```

5 Quit

Finally, the user can select to exit from RATIONALE. This is state 16 which simply ends the execution of RATIONALE.

```
get_action(Next,Userlevel,Domain,SUBP,SYMPTOMS,
           Refinements,Sibling_Refinement,Alternate,AlternateR,
           Enable_mode):-
    Next = 5,
    prompt(["Thank you for using RATIONALE running the 'Domain,'
           workstation problem advisor.]).
```

4.4 Explaining the Selected Problem

State 15 is the hypothesis refinement state. Two main actions are performed in this state. The first of these is problem refinement and the second is hypothesis explanation. Three predicates influence the problem's refinement. The first is *enable_subproblem* which compares the observed symptoms against those required to enable the problem. Once the problem is enabled it is refined as far along the branch it is on allows. *Refine_refinements* is the main predicate which controls the refinement. The third predicate which affects refinement is *alternate*. This predicate compares the observed symptoms against those required to trigger the alternate of the selected problem. If the alternate problem's symptoms are observed, the alternate problem is refined much as the originally selected problem was.

```

/* state 15 */
/* -----refine subproblem----- */
refine_subproblem(Domain,SUBP,SYMPTOMS,Enable_mode,Refinement,
                  Sibling_Refinement,Alternate,AlternateR):-
  enable_subproblem(Domain,SUBP,SYMPTOMS,Enable_mode,
                    Enabled_SUBP,Enable_Symps),
  refine_selected_or_new(Domain,SUBP,SYMPTOMS,Refinement,
                        Enabled_SUBP,Enable_Symps),
  refine_refinements(Domain,Refinement,SYMPTOMS,
                    Sibling_Refinement),
  alternate(Domain,SUBP,SYMPTOMS,Alternate,AlternateR).

```

Enable the Selected Problem

To enable a problem, the variable *Enable_mode* is first examined. If it has the value *user_selected*, the problem's enabling symptom set is extracted from the hypothesis frame slot and intersected with the observed symptom set. If the degree of match is satisfied, the problem is enabled. If the *Enable_mode* is not *user_selected*, it is *system_deduced* and the enabling conditions do not have to be examined. This implies that the user selected *unknown* as the problem type and RATIONALE helped the user select a *problem-symptom* pair to enable problem refinement.

```

/* -----enable subproblem----- */
enable_subproblem(Domain,SubP,SympS,Enable_mode,
                  Enabled_SUBP,Enable_SympS):-
    Enable_mode = user_selected
    -> check_enabled(Domain,SubP,SympS,Enabled_SUBP,Enable_SympS)
    | true.

check_enabled(Domain,SubP,SympS,Enabled_SUBP,Enable_SympS):-
    enable_check(SubP,SympS)
    -> true
    | what_subGs_exist(Domain,TopGN),
      enable_subproblem(TopGN,Enabled_SUBP,Enable_SympS).

```

To check the *user_selected* problem's enabling conditions, the set of respective enabling symptoms are instantiated from the *enabling_conditions* frame slot. These are then intersected with the symptoms observed by the user. If the intersection satisfies the associated required degree of match, the problem is enabled and refined.

```

/* -----> enable check subproblem
enable_check(SubP,SympS):-
    (enabling_cond(refined_SG(SubP),Req_SympS,Req_percent),
     intersect(SympS,Req_SympS,Observed_enable_symps),
     degree_match(Req_SympS,Req_percent,Observed_enable_symps,Resultant_match),
     Resultant_match >= 0) -> true | fail.

```

Refine an Enabled Problem

If a problem is not enabled by the user's first set of observed symptoms, another is selected with another set of observed symptoms. In either case, once the problem is enabled it is refined by the predicate *refine_SUBP*.

```

/* -----refine selected or new problem----- */
refine_selected_or_new(Domain,SUBP,SYMPTOMS,Refinement,
                       Enabled_SUBP):-
    var(Enabled_SUBP),
    refine_SUBP(Domain,SUBP,SYMPTOMS,Refinement).

refine_selected_or_new(Domain,_,SYMPTOMS,Refinement,
                       Enabled_SUBP,Enable_Symp):-
    append(SYMPTOMS,Enable_Symp,New_Symp),
    refine_SUBP(Domain,Enabled_SUBP,New_Symp,Refinement).

```

If Enabled Problem is Invalid, does it have Special Case Symptoms?

The predicate *refine_SUBP* checks if the observed set of symptoms intersects the invalidating set of symptoms to the required degree. If the problem is thus invalidated, its observed symptom set is intersected with the special case symptom set required to override the invalidation. If the special case symptoms are observed to the required degree, the problem is refinable, otherwise its *Refinement* variable is tracked as *invalidated_not_tolerated*.

```

/* -----refine subproblem----- */
refine_SUBP(Domain,SUBP,SYMPTOMS,Refinement):-
  check_invalidated(Domain,SUBP,SYMPTOMS)
  -> tolerated_test(Domain,SUBP,SYMPTOMS,Refinement)
  !check_refinable(Domain,SUBP,SYMPTOMS,Refinement).

tolerated_test(Domain,SUBP,SYMPTOMS,Refinement):-
  check_tolerated(Domain,SUBP,SYMPTOMS)
  -> check_refinable(Domain,SUBP,SYMPTOMS,Refinement)
  ! Refinement = invalidated_not_tolerated.

```

The enabling, the invalidating, the tolerance and the refinement conditions are all checked in the same manner. Their respective symptoms and required degrees of match are instantiated from the corresponding problem's frame slot. Each set is intersected with the user's observed symptoms and the observed degree of match is compared to the required one. A condition is satisfied if the degrees of match are equal or if the observed degree of match exceeds the required one.

```

/* -----invalidated SUBP ? ----- */
check_invalidated(Domain,SubP,SympS):-
  (invalidate_cond(refined_SG(SubP),Req_SympS,Req_percent),
   intersect(SympS,Req_SympS,Observed_invalidate_symps),
   degree_match(Req_SympS,Req_percent,Observed_invalidate_symps,
                Resultant_match),
   Resultant_match >= 0)
  -> true
  ! fail.

```

In an object definition of a particular domain, there are cases where there may be no special case or tolerance symptoms. Such is the case in the ACME domain. Thus, the *check_tolerated* predicate looks for *none* as a possible value for the tolerance symptom set. If *none* is found to be true, then the problem has no special case symptoms to override the invalidating ones and as a consequence once a problem is invalidated, it cannot be refined.

```

/* -----tolerated SUBP ? ----- */
check_tolerated(Domain,SubP,SympS):-
  tolerate_cond(refined_SG(SubP),Req_SympS,Req_percent),
  no_tolerate_symps(Req_SympS,Tolerance_Result),
  (Tolerance_Result = no_tolerance_symps
   -> fail
   | intersect(SympS,Req_SympS,Observed_tolerate_symps),
     degree_match(Req_SympS,Req_percent,Observed_tolerate_symps,Resultant_match),
     Resultant_match >= 0
   -> true
   | fail
  ).

no_tolerate_symps(Req_SympS,Tolerance_Result):-
  Req_SympS = [none]
  -> Tolerance_Result = no_tolerance_symps
  | Tolerance_Result = tolerance_symps.

```

Focus Validated Problem

Once a problem is enabled and not invalidated, it can be refined. As in the case of the tolerance condition, some problems have no refinements and thus, no refinement symptoms. If the refinement slot holds the value *no_refine_to_subG* instead of a set of refinement problem names, the variable *Refinements* passed back to RATIONALE holds

the value *no_refinement_subgoals*. RATIONALE tracks the reasoning explicitly as such in order to explain to the user both the successes and the failures in focusing a problem. If a problem has a set of refinements, it is refined using the predicate *multiple_refinement* otherwise it is refined using *single_refinement*.

```

/* -----refine problem ? ----- */
check_refinable(Domain,SubP,SympS,Refinement):-
  refine_info(refined_SG(SubP),Refine_to,Req_SympS,Req_percent),
  refinable(Refine_to)
  -> refinable_list(SubP,SympS,Refine_to,Req_SympS,Req_percent,
    Refinement),
    check_ref_list(Refinement,EditedR),
    prompt_tree_fragment(SubP,EditedR)
    | Refinement = no_refinement_subgoals.

/* -----> refine info
/* generates the refinement subproblem, its required symptoms and degree of match */
refine_info(refined_SG(SubP),Refine_to,Req_SympS,Req_percent):-
  send(refined_SG(SubP),refinement_subG(Refine_to,Req_SympS,Req_percent)).

/* -----> is it refinable ?
refinable(Refine_to):-
  Refine_to \== no_refine_to_subG,
  Refine_to \== no_ref_symps_observed.

/* -----> is it a set of refinements ?
refinable_list(SubP,SympS,Refine_to,Req_SympS,Req_percent,Refinement):-
  is_list(Refine_to)
  -> multiple_refinement(SubP,SympS,Refine_to,Req_SympS,Req_percent,Refinement)
  | single_refinement(SubP,SympS,Refine_to,Req_SympS,Req_percent,Refinement).

```

The *multiple_refinement* predicate recurses through the list of possible problem refinements. It intersects the sets of observed and required refinement symptoms for each of the problem's refinements. If the intersection is a null set, the atom *no_ref_symps_observed* is appended to the refinement list instead of the name of the refinement subgoal. If the required degree of match is satisfied, the refinement problem is appended to the list *Refinement*. If the required degree of match is not satisfied, the atom

not_enough_ref_symps is appended to the list. As pointed out above, such tracking is used to explain to the user the reasons for any failure in expected refinement.

```

/* -----multiple refinement ----- */
multiple_refinement(_,_,[RIRs]):-!.
multiple_refinement(SubP,Symps,[PossR|PossRs],[ReqS|ReqSs],[ReqP|ReqPs],
    Refinement):-
    refinement_degree_check(Symps,ReqS,ReqP,ObsRefS,Resultant_match),
    (ObsRefS = []
    -> add_element(no_ref_symps_observed,Refinement,[RIRs])
    | Resultant_match >= 0
    -> add_element(PossR,Refinement,[RIRs]),
    | otherwise
    -> add_element(not_enough_ref_symps,Refinement,[RIRs])),
    multiple_refinement(SubP,Symps,PossRs,ReqSs,ReqPs,[RIRs]).

```

The *single_refinement predicate* is used when a problem has only a single sibling. The existence of this predicate is purely an implementation consideration. Recursion is a very slow operation in the implementation of Quintus Prolog for the Xerox 1109 workstation. The *single_refinement* predicate is included to speed up the refinement process in RATIONALE but can be performed as part of the *multiple_refinement* predicate in a different implementation.

```

/* -----single refinement ----- */
single_refinement(SubP,Symps,PossR,Req_S,Req_P,Refinement):-
    refinement_degree_check(Symps,Req_S,Req_P,ObsRefS,Resultant_match),
    (ObsRefS = []
    -> Refinement = [no_ref_symps_observed]
    | Resultant_match >= 0
    -> Refinement = [PossR]
    | otherwise
    -> Refinement = [not_enough_ref_symps]).

```

The focusing of the problem refinements can be achieved in a manner identical to the focusing of the problem itself. The *refine_refinements* predicate checks the value of the *Refinement* variable. If the value of *Refinement* is *no_refinement_subgoals* or *invalidated_not_tolerated*, no further refinement is attempted. Otherwise, the contents of

the variable, *Refinement*, are focused using the *refine_SUBP* predicate recursively if *Refinement* a list or singularly if it represents a single problem.

```

/* -----refine siblings of problem----- */
refine_refinements(Domain,Refinement,SYMPTOMS,Sibling_Refinement):-
  ( (Refinement = no_refinement_subgoals ;
    Refinement = invalidated_not_tolerated)
    -> Sibling_Refinement = no_subGs_no_sibling
  | atom(Refinement)
    -> check_ref_list(Refinement,Refs),
        refine_SUBP(Domain,Refs,SYMPTOMS,Sibling_Refinement)
  | otherwise
    -> check_ref_list(Refinement,Refs),
        refine_refs(Domain,Refs,SYMPTOMS,Sibling_Refinement) ).

/* -----> refine a list of refinements
refine_refs(_,[_],[_]) :- !.
refine_refs(Domain,[R|Rs],SYMPTOMS,Sibling_Refinement):-
  refine_SUBP(Domain,R,SYMPTOMS,SRef),
  add_element(SRef,Sibling_Refinement,[SR|SRs]),
  refine_refs(Domain,Rs,SYMPTOMS,[SR|SRs]).

```

Focus the Problem Alternate for the Valid Problem

After a problem has been focused as far as it can be, its alternate problem slot is examined. If the slot has the value *none*, the Alternate problem variable is assigned *no_alternate* and the original problem's refinement is ended. If however, an alternate problem does exist, the required alternate symptom set is intersected with the original problem's observed symptom set. If the intersection satisfies the required degree of match for the alternate problem condition, the alternate problem is refined in a manner identical to the refinement of the original problem.

```

/* -----alternate ? ----- */
alternate(Domain,SUBP,SYMPTOMS,Alternate,AlternateR):-
    alternate_info(refined_SG(SUBP),Alternate,Req_SympS,Req_percent),
    check_alternate(Domain,SUBP,SYMPTOMS,Alternate,Req_SympS,Req_percent,
        AlternateR).

/* -----> alternate object info
alternate_info(refined_SG(SubP),Alternate,Req_SympS,Req_percent):-
    send(refined_SG(SubP),alternate_subG(Alternate,Req_SympS,Req_percent)).

/* -----> is there an alternate ? is it a list of alternates?
check_alternate(Domain,SubP,SympS,Alternate,Req_SympS,Req_percent,AlternateR):-
    (no_alternate(Alternate)
    -> AlternateR = no_alternate
    ! atom(Alternate)
    -> enable_alternate(Domain,SubP,SympS,Alternate,Req_SympS,Req_percent,
        AlternateR)
    ! otherwise
    -> multiple_alternate(Domain,SubP,SympS,Alternate,Req_SympS,Req_percent,
        AlternateR)).

```

Once the original problem is determined to have an *Alternate*, it is refined using the predicate *enable_alternate* singularly if there is only a single alternate. If the *Alternate* problem is a list, it is refined recursively. The *enable_alternate* predicate performs the *required-to-observed* alternate symptom set intersection and calls *check_refinable* if the required degree of match is satisfied. If the required degree of match is not satisfied, it returns the value *alternate_not_enabled*. Once the alternate is enabled, its refinements and subsequent sibling refinement is performed as described for the user's originally selected problem above.

```

/* ----- is alternate enabled ? ----- */
enable_alternate(Domain,SubP,SympS,Alternate,
    Req_SympS,Req_percent,AlternateR):-
    intersect(SympS,Req_SympS,Observed_alternate_symps),
    degree_match(Req_SympS,Req_percent,
        Observed_alternate_symps,Resultant_match),
    (Resultant_match >= 0)
    -> check_refinable(Domain,Alternate,SympS,AlternateR)
    ! AlternateR = alternate_not_enabled.

```

```

/* -----> recurse 'is alternate enabled' ? ----- */
multiple_alternate(_____,[_____,[AR|ARs]]:- !.
multiple_alternate(Domain,SubP,SympS,[A|As],[RS|RSs],[RPIRPs],AlternateR):-
  enable_alternate(Domain,SubP,SympS,A,RS,RP,AR),
  add_element(AR,AlternateR,[AR|ARs]),
  multiple_alternate(Domain,SubP,SympS,As,RSs,RP,[AR|ARs]).

```

Generate Advice Explaining the Valid Problem and its Refinements

After refining the selected problem and its alternate, RATIONALE generates a set of explanations which are output to the user. These can be viewed as diagnostic recommendations that explain the observed symptoms and suggest to the user various plans of action. The main predicate, *generate_explanations* is the one which extracts the explanations from the respective problem slot. This predicate in turn calls both, *collect_SUBP_E* to generate the selected problem's explanations and *collect_alternate_E* to generate the alternate problem's explanations.

```

/* ----- explanation generator ----- */
generate_explanations(Userlevel,SUBP,Refinements,Sibling_Refinement,
Alternate,AlternateR):-
  collect_SUBP_E(Userlevel,SUBP,Refinements,Sibling_Refinement),
  collect_alternateE(Userlevel,SUBP,Alternate,AlternateR).

```

The explanation collector outputs to the user the most refined explanations it can generate. Thus, if a problem has no refinements, only the originally selected problem's explanations are output. If however, the problem has a set of refinements, the set of refinement explanations preclude the problem's explanations and are output. Similarly, if the refinements of the originally selected problem have their own sibling refinements, then

the explanations associated with the sibling refinements preclude all the higher-level explanations and are subsequently output to the user.

The predicate *collect_SUBP_E* checks if there are no sibling refinements and calls *collect_refineE* if such is the case. If there are sibling refinements, it calls *collect_sibling_refineE*.

```
/* ----- refinement or sibling explanations ? ----- */
collect_SUBP_E(Userlevel,SUBP,Refinements,Sibling_Refinement):-
  (atom(Refinements),no_refs_no_siblings(Refinements,Sibling_Refinement))
  -> prompt(['Your ',SUBP,' can be focused to give the following explanations: ']),
    collect_refineE(Userlevel,SUBP,Refinements)
  | prompt(['Your ',SUBP,' can be FOCUSED to give the following explanations:']),
    collect_sibling_refineE(Userlevel,SUBP,Refinements,Sibling_Refinement).
```

If there are no refinements of the original problem, the predicate *collect_refineE* calls *only_subproblemE* to generate the highest level explanations of the user's problem. Otherwise, it calls *collect_explanations* to generate the set of explanations for the problem's refinement.

```
/* ----- problem or refinement explanations ? ----- */
collect_refineE(Userlevel,SUBP,Refinements):-
  (no_refinements(Refinements)
  -> only_subproblemE(Userlevel,SUBP,SUBP_E)
  | check_ref_list(Refinements,Refs),Refs \== []
  -> collect_explanations(Userlevel,Refs,Refinement_E),
    prompt(['The ',SUBP,' indicates that: ']),
    writeList(Refinement_E)
  | otherwise
  -> only_subproblemE(Userlevel,SUBP,SUBP_E)).

only_subproblemE(Userlevel,SUBP,SUBP_E):-
  get_explanation(Userlevel,SUBP,SUBP_E),
  nl,nl,prompt(['The ',SUBP,' indicates that: ']),
  nl,nl,write(SUBP_E).
```

The *collect_explanations* predicate calls *get_explanation* to instantiate the explanations from their respective frame slots. The explanations are generated on the basis

of the user level. To generate the explanation, a message containing the *Userlevel* and the problem name is sent to the object facility. The facility returns the explanation slot.

```

/* ----- explanation collector ----- */
collect_explanations(_,[],[E!Es]):!.
collect_explanations(Userlevel,[Problem|Problems],Explanations):-
    nonvar(Problem)
    -> (get_explanation(Userlevel,Problem,Explanation),
        add_element(Explanation,Explanations,[E!Es])),
    collect_explanations(Userlevel,Problems,[E!Es]).

get_explanation(Userlevel,Prob,Explanation):-
    Userlevel = novice
    -> send(refined_SG(Prob),novice_explanation(Explanation))
    (Userlevel = experienced
    -> send(refined_SG(Prob),experienced_explanation(Explanation))
    | otherwise
    -> send(refined_SG(Prob),expert_explanation(Explanation)),
    write(Explanation),!.

```

The predicate *collect_sibling_refineE* is structured in a manner similar to *collect_refineE*. The predicate checks that there are refinement siblings that need to be explained by checking for the failure tracking information that RATIONALE collects while refining. Once it is determined that there is a sibling to be explained, the predicate calls *collect_refineE* with the refinement problem as the main problem and its sibling as requiring an explanation. If there is a list of sibling refinements, the above is repeated recursively.

```

/* ----- sibling refinement explanations ----- */
collect_sibling_refineE(Userlevel,SUBP,Refinements,Sibling_Refinement):-
  ((atom(Refinements),Refinements \== [],
    no_refs_no_siblings(Refinements,Sibling_Refinement)),
  -> only_subproblemE(Userlevel,SUBP,SUBP_E)
  | atom(Refinements),Refinements \== []
  -> collect_refineE(Userlevel,Refinements,Sibling_Refinement)
  | otherwise
  -> fail).

collect_sibling_refineE(_,[],_):-!.
collect_sibling_refineE(Userlevel,SUBP,[RIRs],[SRISRs]):-
  check_sib_list(R)
  ->
  collect_refineE(Userlevel,R,SR),
  collect_sibling_refineE(Userlevel,SUBP,Rs,SRs)
  | collect_sibling_refineE(Userlevel,SUBP,Rs,SRs).

```

The alternate explanations are collected using the refinement predicate explanation collectors. If there is no alternate, no alternate explanation is output. If there is a single alternate, the *collect_refineE* predicate is called, otherwise the *collect_sibling_refineE* is called to generate explanations for the alternate problem and its refinements.

```

/* ----- alternate problem explanation collector ----- */
collect_alternateE(Userlevel,SUBP,Alternate,AlternateR):-
  check_no_alternate(AlternateR),
  -> true
  | prompt_alternate(SUBP,Alternate),
  explain_alternate(Userlevel,Alternate,AlternateR).

prompt_alternate(SUBP,Alternate):-
  atom(Alternate)
  -> nl,nl,prompt(['Your ',SUBP,' has an alternate - a ',Alternate,':'])
  | nl,nl,prompt(['Your ',SUBP,' has the following alternates: ']),
  writeList(Alternate).

explain_alternate(Userlevel,Alternate,AlternateR):-
  atom(Alternate)
  -> collect_refineE(Userlevel,Alternate,AlternateR)
  | collect_sibling_refineE(Userlevel,Alternate,AlternateR).

```

4.5 Explaining the Reasoning Strategies

The user asks for explanations about a selected problem's reasoning strategies in state 14. The user is brought to this state by selecting *4. ask questions about session results* from RATIONALE's main menu. State 14 places the user in the session explanation menu. This second menu offers four types of explanations. The first two are *event* and *hypothetical* explanation options. These allow the user to select questions about the refinement of the originally selected problem. The knowledge used to answer these questions is the historical or dynamic knowledge of the session. The second two types of explanation use the static domain knowledge, they are *ability* and *factual* explanation options.

The predicate *sessionE* controls the generation of explanation in state 14. It first displays a menu of the questions using the predicate *display_questions* and then prompts the user for a selection. Once a selection is made, *get_chosen_E* is called to generate the respective explanation.

```
/* state 14 */
/* -----explain session results----- */

sessionE(Userlevel,Domain,SUBP,SYMPTOMS,Refinements,
        Sibling_Refinement,Alternate,AlternateR,Enable_mode):-
    display_questions(SUBP),
    prompt(['Enter your explanation number choice or enter QUIT to return to the
            main menu.']),
    prompt_for_user_selection(EChoice),
    get_chosenE(EChoice,Userlevel,Domain,SUBP,SYMPTOMS,Refinements,
              Sibling_Refinement,Alternate,AlternateR,Enable_mode),
    sessionE(Userlevel,Domain,SUBP,SYMPTOMS,Refinements,
            Sibling_Refinement,Alternate,AlternateR,Enable_mode).
```

4.5.1 Display Questions

The *display_questions* predicate calls *define_Q_lists* which asserts the menu options associated with each type of explanation. It also calls each of the predicates that display the questions associated with the four explanation types.

```
/* -----DISPLAY QUESTIONS----- */
display_questions(SUBP):-
    define_Q_lists(Eventlist,HypoList,AbilityList,FactualList),
    prompt(['The following are the session explanation questions that
           RATIONALE can answer. Simply input the number associated
           with the question and input the specified information you are
           prompted for.']),
    prompt(['EVENT QUESTIONS:']),
    display_event_questions(Eventlist,SUBP),
    prompt(['HYPOTHETICAL QUESTIONS:']),
    display_hypothetical_questions(HypoList,SUBP),
    prompt(['ABILITY QUESTIONS:']),
    display_ability_questions(AbilityList),
    prompt(['FACTUAL QUESTIONS:']),
    display_factual_questions(FactualList).
```


4.5.1.1 Event Questions

The first set of explanation questions are generated using *display_event_questions*. This predicate recurses through the numeric list of event questions and displays their respective text with the current context instantiated as the selected problem context.

```

/* -----EVENT QUESTIONS----- */
display_event_questions([],_):-!.
display_event_questions([EventQ|EventQs],SUBP):-
  \ get_event_Q(EventQ,SUBP),
  display_event_questions(EventQs,SUBP).

get_event_Q(EventQ,SUBP):-
  EventQ = 1,
  prompt(['1 For my ',SUBP,', WHY do you need the symptoms I'm
specifying below?']).

get_event_Q(EventQ,SUBP):-
  EventQ = 2,
  prompt(['2 For my ',SUBP,', WHY do you NOT need the symptoms I'm
specifying below?']).

get_event_Q(EventQ,SUBP):-
  EventQ = 3,
  prompt(['3 For my ',SUBP,', WHY did you conclude the ',SUBP,',
or the subproblem I'm specifying below?']).

get_event_Q(EventQ,SUBP):-
  EventQ = 4,
  prompt(['4 For my ',SUBP,', WHY did you NOT conclude the subproblem
I'm specifying below?']).

get_event_Q(EventQ,SUBP):-
  EventQ = 5,
  prompt(['5 For my ',SUBP,', HOW did you deduce the ',SUBP,',
or the subproblem I'm specifying below?']).

```

4.5.1.2 Hypothetical Questions

The second set of questions displayed in the menu are the hypothetical explanation questions. Like the event questions, their current context is instantiated to the refined problem. The list of hypothetical questions is recursed through using *display_hypothetical_questions* which in turn calls *get_hypothetical_Q* to display each of the questions.

```
/* -----HYPOTHETICAL QUESTIONS----- */
display_hypothetical_questions([],_):-!.
display_hypothetical_questions([HypoQ|HypoQs],SUBP):-
    get_hypothetical_Q(HypoQ,SUBP),
    display_hypothetical_questions(HypoQs,SUBP).

get_hypothetical_Q(HypoQ,SUBP):-
    HypoQ = 6,
    prompt(['6 What-if the symptoms I'm specifying were added to my previous
symptoms for the ',SUBP,'?']).

get_hypothetical_Q(HypoQ,SUBP):-
    HypoQ = 7,
    prompt(['7 What-if the subproblem I'm specifying replaced the ', SUBP, '
with the symptoms I'm also specifying below?']).
```

4.5.1.3 Ability and Factual Questions

Both the ability and factual questions are generated in a manner similar to that of generating the event and hypothetical questions above. Their respective list of questions is

recursed through with the predicates *get_ability_Q* and *get_factual_Q* displaying the respective questions. The main difference is that no context needs to be established for these two types of questions versus the previous two.

```

/* -----ABILITY QUESTIONS----- */
display_ability_questions([]):-!.
display_ability_questions([AbilityQ|AbilityQs):-
    get_ability_Q(AbilityQ),
    display_ability_questions(AbilityQs).

get_ability_Q(AbilityQ):-
    AbilityQ = 8,
    prompt(['8 How do you deduce the subproblem I'm specifying below?']).

get_ability_Q(AbilityQ):-
    AbilityQ = 9,
    prompt(['9 How do you deduce the alternate subproblem for the subproblem
            I'm specifying below?']).

get_ability_Q(AbilityQ):-
    AbilityQ = 10,
    prompt(['10 How do you refine the subproblem I'm specifying below?']).

/* -----FACTUAL QUESTIONS----- */
display_factual_questions([]):-!.
display_factual_questions([FactualQ|FactualQs):-
    get_factual_Q(FactualQ),
    display_factual_questions(FactualQs).

get_factual_Q(FactualQ):-
    FactualQ = '11a',
    prompt(['11a What subproblems can the system handle?']).

get_factual_Q(FactualQ):-
    FactualQ = '11b',
    prompt(['11b What subproblems can the problem I'm specifying be focused
            to?']).

get_factual_Q(FactualQ):-
    FactualQ = 12,
    prompt([' What symptoms and respective degrees of match are']),
    prompt(['12a the relevant          12b /the enabling
            12c /the invalidating      12d /the special case
            12e /the refinement        12f /the alternate ']),
    prompt([' symptoms and degrees of match for the subproblem I'm specifying
            below.']).

```

4.5.2 Generate Explanations

Upon selecting a question from the explanation option menu, the *get_chosenE* predicate calls the corresponding type of explanation generator. The explanation generators are simply predicates that hold strings of texts with variables that are instantiated from the current context. Such a method of explanation is known as template explanation.

4.5.2.1 Event Explanations

Event Explanation for Question 1

When a question is an event explanation type, the predicate *event_explanation* is called. For the first question, *For my Problem, WHY do you need the symptoms I'm specifying below?* the explanation template is instantiated by first generating the symptoms relevant to the current problem, **Problem**, using the predicate *get_relevant_symps*. These are output to help the user select symptoms relevant to the current context. A subset of relevant symptoms is subsequently selected to be explained. Once the symptoms are selected, RATIONALE echoes the user's original questions with the symptoms substituted in. This is one of the methods by which RATIONALE assures the user that their question has been understood. The set of symptoms to be explained is

passed to the *explanation_collector_Q1* predicate. Examples of RATIONALE's event explanations for the ACME domain are included in Appendix B2.

```
/* -----EVENT EXPLANATION TEMPLATE 1----- */
event_explanation(QuestionN,ExplainSUBP,Userlevel):-
    QuestionN = 1,
    get_relevant_symps(ExplainSUBP),
    prompt_sympoms,
    prompt(['WHY do you need these symptoms for my ',ExplainSUBP,': ']),
    get_sympoms(ExplainSUBP,ReqSymps),
    get_event_Q(QuestionN,ExplainSUBP,ReqSymps),
    prompt(['Your symptoms are needed since:']),
    explanation_collector_Q1(ExplainSUBP,ReqSymps,Userlevel).
```

The explanation collector for the template 1 checks the role that the selected symptoms played in the problem refinement. It checks if they were enabling, invalidating, tolerance, refinement or alternate problem symptoms by calling the respective condition collector predicate.

```
/* -----template 1 explanation collector----- */
explanation_collector_Q1(ExplainSUBP,ReqSymps,Userlevel):-
    enable_collector_Q1(ExplainSUBP,ReqSymps),
    invalidate_collector_Q1(ExplainSUBP,ReqSymps),
    tolerate_collector(ExplainSUBP,ReqSymps),
    refinement_collector_Q1(ExplainSUBP,ReqSymps,Userlevel),
    alternate_collector_Q1(ExplainSUBP,ReqSymps,Userlevel).
```

The *enable_collector_Q1* predicate intersects the current context symptoms with those that the user wishes explained. If the intersection is not null and the user wishes enabling symptoms to be explained, a string of text with the instantiated enabling symptoms and the respective current problem is output.

```

/* -----template 1 enabling condition collector----- */
enable_collector_Q1(ExplainSUBP,ReqSymps):-
    enable_lookup(ExplainSUBP,EnableS,_),
    intersect(ReqSymps,EnableS,EShareS),
    EShareS \== []
    -> prompt(['--> the symptom set ']),
        write(EShareS),
    prompt([' enables 'ExplainSUBP,' so that it may be focused for an
            explanation.'])
    ! true.

```

The *invalidate_collector_Q1* and the *tolerate_collector_Q1* predicates are similar to the enabling one. In each predicate, the user's selected symptoms are intersected with those required to invalidate or tolerate the problem. If the intersection is again not null, a string of text explaining the instantiated problem and its respective invalidating or tolerance symptoms is output to the user.

```

/* -----template 1 invalidation condition collector----- */
invalidate_collector_Q1(ExplainSUBP,ReqSymps):-
    invalidate_lookup(ExplainSUBP,InvalidS,_),
    intersect(ReqSymps,InvalidS,IShareS),
    IShareS \== []
    -> prompt(['--> the symptom set ']),
        write(IShareS),
    prompt([' invalidates 'ExplainSUBP,' so that other observed symptoms
            may be examined for special case conditions.'])
    ! true.

/* -----template 1 tolerance condition collector----- */
tolerate_collector(ExplainSUBP,ReqSymps):-
    tolerate_lookup(ExplainSUBP,TolerateSymps,_),
    intersect(ReqSymps,TolerateSymps,TShareS),
    TShareS \== []
    -> prompt(['--> the symptom set ']),
        write(TShareS),
    prompt([' provides special conditions for 'ExplainSUBP,' so that it may
            override other observed symptoms that may invalidate it.'])
    ! true.

```

The *refinement_collector_Q1* predicate is more complex than the enabling, invalidating and tolerance explanation predicates. If the problem has a single refinement, it is not explained in the same manner as if it had several refinements. Thus, two predicates are used to generate the explanation templates depending on which case is to be explained.

```

/* -----template 1 refinement condition collector----- */
refinement_collector_Q1(ExplainSUBP,ReqSymps,Userlevel):-
    refinement_lookup(ExplainSUBP,RefineE,RefineS,_),
    atom(RefineE),
    single_refine_collector_Q1(ExplainSUBP,RefineE,RefineS,ReqSymps,
                               Userlevel).

refinement_collector_Q1(ExplainSUBP,ReqSymps,Userlevel):-
    refinement_lookup(ExplainSUBP,RefineE,RefineS,_),
    multiple_refine_collector_Q1(ExplainSUBP,RefineE,RefineS,ReqSymps,
                                  Userlevel).

```

The predicate *single_refine_collector_Q1* checks that the current problem has a refinement subgoal and intersects the corresponding refinement symptoms with those symptoms the user selected for an explanation. If the intersection is not null, the predicate outputs a string of text explaining the relation of the selected symptom to the refinement of the current problem. The single refinement explanation predicate uses the *within_level* predicate to determine whether the level of refinement should be included in the explanation. If it is not to be included according to the user level, RATIONALE simplifies the explanation by relating the refinement symptoms only to the current problem rather than relating them to both the current problem and its refinement subgoal.

```

/* -----template 1 single refinement collector----- */
single_refine_collector_Q1(ExplainSUBP,RefineE,RefineS,ReqSymps,
                           Userlevel):-
    (( within_level(Userlevel,RefineE),
        RefineE \== no_refine_to_subG,
        intersect(ReqSymps,RefineS,RShareS),
        RShareS \== [] )
    -> prompt(['--> the symptom set ']),
        write(RShareS),
        prompt([' is a focusing symptom set for the 'ExplainSUBP,' so that it
                may be focused to a 'RefineE;'.'])
    | RefineE \== no_refine_to_subG,
        intersect(ReqSymps,RefineS,RShareS),
        RShareS \== []
    -> prompt(['--> the symptom set ']),
        write(RShareS),
        prompt([' is a focusing symptom set for the 'ExplainSUBP,' used to direct
                the system to the source of your problem.'])
    | otherwise
    -> true).

```


problem to be explained. If the problem *Complexity* is less than the maximum detail, then the problem can be included in the explanation. If the *Complexity* is greater than the maximum allowed detail, the relative importance of including the problem in the explanation is calculated. The relative importance is calculated by subtracting a pre-set number (2 in RATIONALE's case), from the *Importance* value attached to the problem. If the calculated relative importance is found to be greater than the maximum allowed detail, the problem is included in the explanation, otherwise it is not. This technique of calculating relative complexity and importance is based on work by Wallis and Shortliffe described in [Wallis82].

```

/* -----find complexity importance ranking ----- */
within_level(Userlevel,Prob):-
  get_max_level(Userlevel,Max_Detail),
  explanation_detail_lookup(Prob,Complexity,Importance),
  within_detail(Max_Detail,Complexity,Importance).

get_max_level(Userlevel,Max_Detail):-
  get_user_rating(Userlevel,Expertise),
  get_user_detail(Userlevel,Detail),
  Max_Detail is (Expertise + Detail).

get_user_rating(Userlevel,Expertise):-
  (Userlevel = novice -> Expertise = 2
   | Userlevel = experienced -> Expertise = 4
   | otherwise -> Expertise = 6).

get_user_detail(Userlevel,Detail):-
  (Userlevel = novice -> Detail = 1
   | Userlevel = experienced -> Detail = 2
   | otherwise -> Detail = 3).

explanation_detail_lookup(Prob,SUBPCComplexity,SUBPImportance):-
  explanation_detail(refined_SG(Prob),SUBPCComplexity,SUBPImportance).

within_detail(Max_Detail,Complexity,Importance):-
  ( Complexity <= Max_Detail
  -> true
  | RelImportance is (Importance - 2) ,
    RelImportance >= Max_Detail
  -> true
  | otherwise -> fail).

```

Collecting the explanations associated with the symptoms required for refining the alternate problem are similar to collecting the explanations required to explain the problem's refinement symptoms themselves. The *single_alternate_collector_Q1* is used to generate explanations for a problem which only has a single alternate problem. The *multiple_alternate_collector_Q1* generates explanations for a problem with several alternate problems.

```
/* -----template 1 alternate condition collector----- */
alternate_collector_Q1(ExplainSUBP,ReqSymps,Userlevel):-
    alternate_lookup(ExplainSUBP,AlternateE,AlternateS,_),
    atom(AlternateE),
    single_alternate_collector_Q1(ExplainSUBP,AlternateE,AlternateS,
                                  ReqSymps,Userlevel).

alternate_collector_Q1(ExplainSUBP,ReqSymps,Userlevel):-
    alternate_lookup(ExplainSUBP,AlternateE,AlternateS,_),
    multiple_alternate_collector_Q1(ExplainSUBP,AlternateE,AlternateS,
                                     ReqSymps,Userlevel).
```

The predicate *single_alternate_collector_Q1* behaves like the *single_refinement_collector_Q1* and checks that the current problem has a single alternate problem and intersects the corresponding required alternate symptoms with those symptoms the user selected for an explanation. If the intersection is not null, the predicate outputs a string of text explaining the relation of the selected symptom to the alternate of the current problem. The single alternate explanation predicate also uses the *within_level* predicate to determine the level of explanation for the alternate problem.

```

/* -----template 1 single alternate collector----- */
single_alternate_collector_Q1(ExplainSUBP,AlternateE,AlternateS,
                               ReqSymps,Userlevel):-
  (( within_level(Userlevel,AlternateE),
    AlternateE \== none,
    intersect(ReqSymps,AlternateS,AShareS),
    AShareS \== [] )
  -> prompt(['--> the symptom set ']),
    write(AShareS),
    prompt([' is an alternate problem symptom set for the 'ExplainSUBP,' so
            that it may be focused to a ',AlternateE,'])
  | AlternateE \== none,
    intersect(ReqSymps,AlternateS,AShareS),
    AShareS \== []
  -> prompt(['--> the symptom set ']),
    write(AShareS),
    prompt([' is a focusing symptom set for the 'ExplainSUBP,' used to direct
            the system to a problem that may be the source of your
            problem.'])
  | otherwise
  -> true).

```

The *multiple_alternate_collector_Q1* predicate behaves in a manner similar to *multiple_refinement_collector_Q1* and recurses through the list of alternates for the current problem. For each alternate, it checks the intersection between the symptoms selected for explanation and the required alternate symptoms, if the intersection is not null, it outputs the string of explanation text instantiated according to the current context. It again selects the explanation text appropriate to the user level using the predicate *within_detail*.

```

/* -----template 1 multiple alternate collector----- */
multiple_alternate_collector_Q1(.,[],[],_):-!.
multiple_alternate_collector_Q1(ExplainSUBP,[AE|AEs],[AS|ASs],
                                ReqSymps,Userlevel):-
  ( (within_level(Userlevel,AE),
    intersect(ReqSymps,AS,AShareS),
    AShareS \== [] )
  -> prompt(['--> the symptom set ']),
    write(AShareS),
    prompt([' is an alternate problem symptom set for the 'ExplainSUBP,' so
      that it may be focused to a 'AE,').]),
    multiple_alternate_collector_Q1(ExplainSUBP,AEs,ASs,ReqSymps,
                                    Userlevel)
  | within_level(Userlevel,AE),
    intersect(ReqSymps,AS,AShareS),
    AShareS \== []
  -> prompt(['--> the symptom set ']),
    write(AShareS),
    prompt([' is a focusing symptom set for the 'ExplainSUBP,' used to direct
      the system to a problem that may be the source of your problem.']),
    multiple_alternate_collector_Q1(ExplainSUBP,AEs,ASs,ReqSymps,
                                    Userlevel)
  | otherwise
  -> multiple_alternate_collector_Q1(ExplainSUBP,AEs,ASs,ReqSymps,
                                    Userlevel)).

```

Event Explanation for Question 2

For the second event explanation question, *For my Problem, WHY do you NOT need the symptoms I'm specifying below?*, the explanation is generated by a second clause of the *event_explanation* predicate. The predicate first looks up the relevant symptoms associated with the current problem. It intersects this set with the set of originally observed symptoms used to refine the current problem. The resultant set is output to the user as the relevant symptoms that were not used to refine the selected problem. The user is then asked to select a set of these unused symptoms. Once the

selection is made the predicate *explanation_collector_Q2* is called to generate the explanation of why the selected symptoms were not used.

```

/* -----EVENT EXPLANATION TEMPLATE 2----- */
event_explanation(QuestionN,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
                ERefinements,ESiblings,EAltemate,EAltemateR):-
    QuestionN = 2,
    relevant_lookup(ExplainSUBP,RelSymps),
    subtract(RelSymps,ExplainSYMPTOMS,NotSelSymps),
    prompt(['These symptoms were the relevant symptoms not selected for the
    ',ExplainSUBP,':']),
    writeList(NotSelSymps),
    prompt_sympoms,
    prompt(['WHY do you NOT need these symptoms for my ',ExplainSUBP,': ']),
    get_sympoms(ExplainSUBP,NotReqSymps),
    get_event_Q(QuestionN,ExplainSUBP,NotReqSymps),
    prompt(['Your symptoms are NOT needed since:']),nl,
    explanation_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps,
                            Userlevel,ERefinements,ESiblings,
                            EAltemate,EAltemateR).

```

Note that examples of RATIONALE's explanations are included in Appendix B2. The explanation collector for the second template behaves in a manner very similar to the first. The main difference being that this template is explaining why symptoms were not used. The explanation generated explains why the symptoms were not needed to enable, invalidate or tolerate the selected problem. For applications other than ACME, it may also be useful to explain why symptoms were not used to refine or generate an alternate to the current problem. This can be done easily in a manner similar to the explanation for refinement and alternate symptom use described in template 1 above.

```

/* -----template 2 explanation collector----- */
explanation_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps,
                        Userlevel,ERefines,ESiblings,
                        EAltere,EAltemateR):-
    enable_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps),
    invalidate_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps),
    tolerate_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps).

```

The enable, invalidate and tolerate explanation generators for template 2 all behave in a similar manner. The required symptoms to satisfy the condition are instantiated from their respective frame slot. This set is then intersected with the symptom set the user has selected for an explanation. If the intersection is not null, the originally observed symptom set is also intersected with the set required to satisfy the particular condition. This second intersection is used to explain to the user that there were other symptoms to satisfy the condition and why the symptoms they selected for explanation were not used.

```

/* -----template 2 enabling condition collector----- */
enable_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps):-
    enable_lookup(ExplainSUBP,EnableS,_),
    intersect(EnableS,NotReqSymps,EShareS),
    EShareS \== []
-> intersect(EnableS,ExplainSYMPTOMS,EObsShareS),
    prompt(['--> the symptom set ']),
    write(EShareS),
    prompt([' was not required since the set, ']),
    write(EObsShareS),
    prompt([' enabled ',ExplainSUBP,' so that it may be focused for an
            explanation.'])
! true.

/* -----template 2 invalidating condition collector----- */
invalidate_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps):-
    invalidate_lookup(ExplainSUBP,InvalidS,_),
    intersect(InvalidS,NotReqSymps,IShareS),
    IShareS \== []
-> intersect(InvalidS,ExplainSYMPTOMS,IObsShareS),
    prompt(['--> the symptom set ']),
    write(IShareS),
    prompt([' was not required since the set ',IObsShareS,' invalidated
            ',ExplainSUBP,' which required other observed symptoms to be
            examined for special case conditions.'])
! true.

/* -----template 2 tolerance condition collector----- */
tolerate_collector_Q2(ExplainSUBP,ExplainSYMPTOMS,NotReqSymps):-
    tolerate_lookup(ExplainSUBP,TolerateSymps,_),
    intersect(TolerateSymps,NotReqSymps,TShareS),
    TShareS \== []
-> intersect(TolerateSymps,ExplainSYMPTOMS,TObsShareS),
    prompt(['--> the symptom set ']),
    write(TShareS),

```

```

prompt([' was not required since the set ']),
write(TObsShareS),
prompt([' provided the special conditions for 'ExplainSUBP,' so that it
may override other observed symptoms that invalidated it.'])
! true.

```

Event Explanation for Question 3

The explanation generator for the third event question explains the question, *For my Problem, WHY did you conclude the Problem, or the subproblem I'm specifying below?* The third clause of the *event_explanation* predicate first outputs to the user a list of relevant problems and subproblems concluded for the originally selected problem. Once the user specifies a problem to be explained the predicate *explanation_collector_Q3* is called. (Again, note Appendix B2 for examples).

```

/* -----EVENT EXPLANATION TEMPLATE 3----- */
event_explanation(QuestionN,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
    ERefinements,ESiblings,EAlternate,EAlternateR):-
    QuestionN = 3,
    prompt(['These were the related problems and subproblems that were
concluded for the 'ExplainSUBP]),
    check_R_list(ERefinements,ResultR)
    -> writeList(ResultR),
    check_A_list(EAlternate,ResultA),
    atom(EAlternate)
    -> nl,write(EAlternate),
    prompted_line(['Why did you conclude a: '],SelfPcode),
    name(SelfP,SelfPcode),
    prompt(['RATIONALE was able to conclude a 'SelfP,' since:']),
    explanation_collector_Q3(SelfP,ExplainSUBP,ExplainSYMPTOMS,
        Userlevel,ResultR,ESiblings,EAlternate,EAlternateR).

```

Three clauses of the `explanation_collector_Q3` predicate are used to provide the user with an explanation. The first clause is used if the user selects the originally selected problem as the one to be explained. The explanation is generated by intersecting the required enabling, invalidating and tolerance symptoms with those symptoms originally observed by the user. For each case, the user is told which symptoms, if any, enabled, invalidated or tolerated the problem selected for explanation.

```

/* -----template 3 explanation collector > explain Problem----- */
explanation_collector_Q3(SeIP, ExplainSUBP, ExplainSYMPTOMS,
                        Userlevel, ResultR, ESiblings,
                        EAlternate, EAlternateR):-
    SeIP = ExplainSUBP,
    enable_lookup(ExplainSUBP, EnableS, EnableDeg),
    intersect(ExplainSYMPTOMS, EnableS, EShareS),
    prompt(['--> your symptom set ']),
    write(EShareS),
    prompt([' allowed your ', ExplainSUBP, ' to be enabled to the required degree of
            ', EnableDeg]),
    invalidate_lookup(ExplainSUBP, InvalidS, InvalidDeg),
    intersect(ExplainSYMPTOMS, InvalidS, IShareS),
    IShareS \= [],
    -> nl, prompt(['--> your symptom set ']),
    write(IShareS),
    prompt([' invalidated your ', ExplainSUBP, ' to the required degree of
            ', InvalidDeg]),
    tolerate_lookup(ExplainSUBP, TolerateS, TolerateDeg),
    intersect(ExplainSYMPTOMS, TolerateS, TShareS),
    prompt(['--> your symptom set ']),
    write(TShareS),
    prompt([' provided special case conditions for your ', ExplainSUBP, ' to the
            required degree of ', TolerateDeg, ' which allowed it to override the
            invalidating symptom set and remain enabled.']).

```

If the user selects one of the refinement subproblems to be explained, the second clause of the `explanation_collector_Q3` predicate is used. The refinement symptoms associated with the subproblem are intersected with the observed symptoms.

The resultant set is used to explain to the user the symptoms that satisfied the conditions for the refinement of the original problem.

```
/* -----template 3 explanation collector > explain Refinement----- */
explanation_collector_Q3(SelP,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
                        ResultR,ESiblings,EAlternate,EAlternateR):-
    memberchk(SelP,ResultR),
    refinement_lookup(ExplainSUBP,RefineE,RefineS,RefineDeg),
    correspond(SelP,RefineE,RefineS,SelPS),
    correspond(SelP,RefineE,RefineDeg,SelPDeg),
    intersect(ExplainSYMPTOMS,SelPS,RShareS),
    prompt(['--> your symptom set ']),
    write(RShareS),
    prompt([' allowed your 'ExplainSUBP,' to be focused to a ',SelP,' to match
           the required degree of a minimum of ',SelPDeg]).
```

The third clause of the explanation collector for the event question 3 explains *why* an alternate problem is deduced. This is done in a manner similar to *why* a problem refinement is performed. The required alternate problem symptoms are instantiated from the frame slot and intersected with those observed by the user. The resultant symptom set is output to the user with the required degree of match as the explanation of *why* the alternate problem was deduced.

```
/* -----template 3 explanation collector > explain Alternate----- */
explanation_collector_Q3(SelP,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
                        ResultR,ESiblings, EAlternate,EAlternateR):-
    SelP = EAlternate,
    alternate_lookup(ExplainSUBP,AlternateE,AlternateS,AlternateDeg),
    intersect(ExplainSYMPTOMS,AlternateS,AShareS),
    prompt(['--> your symptom set ']),
    write(AShareS),
    prompt([' allowed your 'ExplainSUBP,' to be focused to an alternate
           problem, a ',SelP,' to match the required degree of ',
           AlternateDeg,'This alternate problem may also be the source of your
           symptoms.']).
```

Event Explanation for Question 4

The fourth event explanation question in the menu of user questions is, *For my Problem, WHY did you NOT conclude the subproblem I'm specifying below?* The *event_explanation* clause prompts the user with the relevant problems that were not concluded for a selection. Examples of these prompts are included in Appendix B2. The *event_explanation* clause calls the *explanation_collector_Q4* to generate the explanation for why the problem was not concluded.

```
/* -----EVENT EXPLANATION TEMPLATE 4----- */
event_explanation(QuestionN,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
                ERefinements,ESiblings,EAlternate,EAlternateR):-
    QuestionN = 4,
    prompt(['These were the related problems and subproblems that were NOT
            concluded for the ',ExplainSUBP]),
    refinement_lookup(ExplainSUBP,RefineE,_,_),
    alternate_lookup(ExplainSUBP,AlternateE,_,_),
    check_R_list(ERefinements,ResultR)
    -> subtract(RefineE,ResultR,NotRef),
    writeList(NotRef),
    include_alternate(EAlternate),
    prompted_line(['Why did you NOT conclude a: ',SelPcode),
    name(SelP,SelPcode),
    prompt(['RATIONALE was NOT able to conclude a ',SelP,' since:']),
    explanation_collector_Q4(SelP,ExplainSUBP,ExplainSYMPTOMS,
                            Userlevel,NotRef,ESiblings,
                            EAlternate,EAlternateR).
```

The *explanation_collector_Q4* predicate generates a template explanation for *why* a refinement of the selected problem is not concluded. The required refinement symptoms are intersected with those observed by the user. The difference between the intersected set and the required set of symptoms is generated and it becomes the needed symptoms. The

absence of these needed symptoms becomes the reason why a particular refinement was not concluded.

```

/* -----template 4 explanation collector > explain Refinement----- */
explanation_collector_Q4(SelP, ExplainSUBP, ExplainSYMPTOMS, Userlevel,
                        NotRef, ESiblings, EAlternate, EAlternateR):-
    memberchk(SelP, NotRef),
    refinement_lookup(ExplainSUBP, RefineE, RefineS, RefineDeg),
    correspond(SelP, RefineE, RefineS, SelPS),
    correspond(SelP, RefineE, RefineDeg, SelPDeg),
    intersect(ExplainSYMPTOMS, SelPS, RShareS),
    subtract(SelPS, RShareS, NeedS),
    prompt_Ref_Q4(RShareS, ExplainSUBP, ExplainSYMPTOMS, SelP,
                  SelPDeg, NeedS).

prompt_Ref_Q4(RShareS, ExplainSUBP, ExplainSYMPTOMS, SelP,
              SelPDeg, NeedS):-
    RShareS \== [],
    prompt(['--> your symptom set ']),
    write(RShareS),
    prompt([' did NOT allow your ', ExplainSUBP, ' to be focused to a ', SelP, ' To
            match the required degree of a minimum of ', SelPDeg, ' you also
            needed to observe the set, ']),
    write(NeedS).

prompt_Ref_Q4(RShareS, ExplainSUBP, ExplainSYMPTOMS, SelP, SelPDeg,
              NeedS):-
    prompt(['--> your observed symptom set, ']),
    write(ExplainSYMPTOMS),
    prompt([' did NOT allow your ', ExplainSUBP, ' to be focused to a ', SelP, ' To
            match the required degree of a minimum of ', SelPDeg, ' you also
            needed to observe the set, ']),
    write(NeedS).

```

The second clause of the *explanation_collector_Q4* predicate generates an explanation of why an alternate problem is not concluded. This is achieved in a manner similar to that of why a refinement problem was not concluded. The alternate problem symptom set is intersected with the observed symptom set. The difference between the resultant intersection and the required alternate symptoms forms the set of missing symptoms needed to conclude the alternate problem.

These symptoms are output to the user as part of the explanation template of why the alternate problem was not concluded.

```

/* -----template 4 explanation collector > explain Alternate----- */
explanation_collector_Q4(SelP,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
                        ResultR,ESiblings,EAlternate,EAlternateR):-
    alternate_lookup(ExplainSUBP,AlternateE,AlternateS,AlternateDeg),
    SelP = AlternateE,
    intersect(ExplainSYMPTOMS,AlternateS,AShareS),
    subtract(AlternateS,AShareS,NeedS),
    prompt_alt_Q4(AShareS,ExplainSUBP,ExplainSYMPTOMS,
                  AlternateE,AlternateS,AlternateDeg,NeedS).

prompt_alt_Q4(AShareS,ExplainSUBP,ExplainSYMPTOMS,
              AlternateE,AlternateS,AlternateDeg,NeedS):-
    AShareS \== [],
    prompt(['--> your symptom set ']),
    write(AShareS),
    prompt([' did NOT allow your ',ExplainSUBP,' to be focused to an alternate
            problem, a ',AlternateE,' To match the required degree of a
            minimum', AlternateDeg,' you also needed to observe the set, ']),
    write(NeedS).

prompt_alt_Q4(AShareS,ExplainSUBP,ExplainSYMPTOMS,
              AlternateE,AlternateS,AlternateDeg,NeedS):-
    prompt(['--> your observed symptom set, ']),
    write(ExplainSYMPTOMS),
    prompt([' did NOT allow your ',ExplainSUBP,' to be focused to an alternate
            ',AlternateE,' To match the required alternate problem degree of a
            minimum of ',AlternateDeg,' you also needed to observe the set, ']),
    write(NeedS).

```

Event Explanation for Question 5

The fifth event question from the menu is *For my Problem, HOW did you deduce the Problem or the subproblem I'm specifying below?* The fifth clause of the *event_question* predicate generates all the relevant refinement and alternate subproblems of the original problem. The user is prompted for a selection from this relevant set of problems. Note Appendix B2 for examples of such explanations. Once a problem is selected, the predicate *decide_collector* is called to generate the template explanation.

```

/* -----EVENT EXPLANATION TEMPLATE 5----- */
event_explanation(QuestionN,ExplainSUBP,ExplainSYMPTOMS,Userlevel,
                 ERefinements,ESiblings,EAlternate,EAlternateR):-
    QuestionN = 5,
    prompt(['These were the related problems and subproblems that were
            concluded for the ',ExplainSUBP]),
    check_R_list(ERefinements,ResultR)
    -> writeList(ResultR),
    flatten(ESiblings,ESib),
    check_R_list(ESib,ResultSib)
    -> writeList(ResultSib),
    check_A_list(EAlternate,ResultA),
    atom(EAlternate)
    -> write(EAlternate),
    check_R_list(EAlternateR,ResultAltR)
    -> writeList(ResultAltR),
    prompted_line(['For my ',ExplainSUBP,' HOW did you conclude a:
                  ',SelPcode),
    name(SelP,SelPcode),
    prompt(['RATIONALE was able to conclude a ',SelP,' since:']),
    decide_collector(SelP,ExplainSUBP,ExplainSYMPTOMS,
                    Userlevel,ResultR,ResultSib,
                    EAlternate,ResultAltR).

```

If the user selects the original problem, a first level refinement or a first level alternate of the original problem, the explanation template of question 3 is used. The question 3 template already provides an explanation for those three cases. If however, the

user wishes a sibling refinement or a sibling of an alternate problem to be explained, the predicate, `explanation_collector_Q5` is called.

```

/* -----template 5 explanation collector > explain Problem, Refinement or Alternate--- */
decide_collector(SelP, ExplainSUBP, ExplainSYMPTOMS, Userlevel, ResultR,
                 ResultSib, EAlternate, ResultAltR):-
    (SelP = ExplainSUBP;
     memberchk(SelP, ResultR);
     SelP = EAlternate),
    explanation_collector_Q3(SelP, ExplainSUBP, ExplainSYMPTOMS,
                             Userlevel, ResultR, ResultSib, EAlternate,
                             ResultAltR).

/* ----template 5 explanation collector > explain Sibling of Refinement or
Alternate ----- */
decide_collector(SelP, ExplainSUBP, ExplainSYMPTOMS,
                 Userlevel, ResultR, ResultSib, EAlternate, ResultAltR):-
    explanation_collector_Q5(SelP, ExplainSUBP, ExplainSYMPTOMS,
                             Userlevel, ResultR, ResultSib,
                             EAlternate, ResultAltR).

```

The sibling of a problem's refinement is explained using the *within_level* predicate. Thus, if explaining a sibling exceeds the maximum level of detail that should be output to a user, then not all the levels traversed to conclude the sibling are described. To explain *why* the sibling was concluded and once the explanation is determined to be within the allowed level of detail, the explanation collector predicate of question 3 is called with both the problem and its first refinement are included in the explanation template. If the level of detail exceeds that which is allowed to the user, the sibling is identified as a refinement of the original problem, without indicating the node the sibling was refined from. A refinement of an alternate problem to that originally selected is explained in a manner similar to that of a sibling refinement. The difference between the two predicates being that one generates a template for a refinement sibling and the other generates a template for an alternate problem refinement.

```

/* -----template 5 explanation collector > explain Sibling----- */
explanation_collector_Q5(SelP,ExplainSUBP,ExplainSYMPTOMS,
    Userlevel,ResultR,ResultSib,EAlternate,ResultAltR):-
    memberchk(SelP,ResultSib),
    sub_goal(refined_SG(SelP),Parent,SelP),
    within_level(Userlevel,Parent)
-> prompt(['--> your 'Parent,' was a focusing problem for your
    'ExplainSUBP]),
    explanation_collector_Q3(SelP,Parent,ExplainSYMPTOMS,
        Userlevel,ResultSib,_,EAlternate,ResultAltR)
| explanation_collector_Q3(SelP,Parent,ExplainSYMPTOMS,Userlevel,
    ResultSib,_,EAlternate,ResultAltR).

```

```

/*-----template 5 explanation collector > explain Alternate-----*/
explanation_collector_Q5(SelP, ExplainSUBP, ExplainSYMPTOMS, Userlevel,
    ResultR, ResultSib, EAlternate, ResultAltR):-
    memberchk(SelP, ResultAltR),
    sub_goal(refined_SG(SelP), Parent, SelP),
    within_level(Userlevel, Parent)
-> prompt(['--> your 'EAlternate,' was an alternate problem for your
    'ExplainSUBP]),
    explanation_collector_Q3(SelP, Parent, ExplainSYMPTOMS,
        Userlevel, ResultAltR, Parent, _)
| explanation_collector_Q3(SelP, Parent, ExplainSYMPTOMS,
    Userlevel, ResultAltR, Parent, ).

```

4.5.2.2 Hypothetical Explanations

The next set of questions that the user can select from the session explanation menu is the hypothetical set. Note Appendix B3 for examples of RATIONALE's hypothetical explanations for the ACME domain.

Hypothetical Explanation for Question 6

The predicate *get_chosenE* calls the *hypothetical_explanation* predicate to answer the user question. The first hypothetical explanation generator answers the question, *WHAT-if the symptoms I'm specifying below were added to my previous symptoms for the Problem?* It first reminds the user of the symptoms that were previously selected and displays the relevant symptoms that the user can add to the previously selected set. Once the user selects the symptoms to be added, RATIONALE generates a hypothetical set of recommendations based on appending the hypothetical symptoms to the original ones. The re-generation of recommendations is done by calling the *problem refinement* module described in section 4.4 above.


```

/* -----HYPOTHETICAL EXPLANATION TEMPLATE 6----- */
hypothetical_explanation(QuestionN, ExplainSUBP, ObsSymptoms, Userlevel):-
    QuestionN = 6,
    prompt(['To remind you, Your previously observed symptoms are:']),
    writeList(ObsSymptoms),
    get_relevant_symps(ExplainSUBP),
    prompt_symptoms,
    prompt(['WHAT-IF these symptoms were added to my above ones for
            'ExplainSUBP,': '']),nl,
    get_symptoms(ExplainSUBP, AddedSymps),
    get_hypothetical_Q(QuestionN, AddedSymps, ExplainSUBP),
    append(AddedSymps, ObsSymptoms, HypoSymptoms),
    prompt(['If these symptoms were added to the previous ones for
            'ExplainSUBP,': '']),
    writeList(AddedSymps),
    explainer(Userlevel, Domain, ExplainSUBP, HypoSymptoms, user_selected).

```

Hypothetical Explanation for Question 7

The explanation template for the seventh question is very similar to that of the sixth hypothetical question. The hypothetical question the user can ask is, *What-if the New Problem replaced the Problem with these new symptoms?* Thus, the user wishes to re-select the problem and its symptoms and generate the associated hypothetical explanations. The explanation template prompts the user for the new problem and its associated symptoms. It does this using the predicates of the refinement algorithm of section 4.4. Once the user selections are made, it calls the refinement algorithm's *explainer* predicate to generate the hypothetical explanations. It should be noted here that the hypothetical problem-information generated by the hypothetical explanation session is kept separate from the originally selected user problem and once RATIONALE answers the user's hypothetical question, the information is discarded.

```

/* -----HYPOTHETICAL EXPLANATION TEMPLATE 7----- */
hypothetical_explanation(QuestionN,ExplainSUBP,_,Userlevel):-
    QuestionN = 7,
    get_problem(HypoSUBP),
    get_relevant_symps(HypoSUBP),
    prompt_symptoms,
    prompt(['WHAT-IF these symptoms are observed for ',HypoSUBP,': ']),
    get_symptoms(HypoSUBP,HypoSymps),
    get_hypothetical_Q(QuestionN,ExplainSUBP,HypoSUBP,HypoSymps),
    explainer(Userlevel,Domain,HypoSUBP,HypoSymps,user_selected).

```

4.5.2.3 Ability Explanations

Ability explanations are the third type of explanations that the user can request. These and the factual explanations use the knowledge defined in the domain to answer user questions about reasoning methods. Note Appendix B4 for examples of RATIONALE's ability explanations.

Ability Explanation for Question 8

The question, *How do you deduce the subproblem I'm specifying below ?*, is one of the ability questions. It is answered by extracting the symptoms required to satisfy the enabling, invalidating and tolerance conditions from the respective problem's frame slots. Once the required symptoms and degrees of match are extracted, a corresponding explanation is generated.

g.

```

/* -----ABILITY EXPLANATION TEMPLATE 8----- */
ability_explanation(QuestionN):-
    QuestionN = 8,
    prompted_line(['
subproblem: '),ExplainSUBP_code),
    name(ExplainSUBP, ExplainSUBP_code),
    get_ability_Q(QuestionN,ExplainSUBP),
    nl,
    enable_lookup(ExplainSUBP,EnableS,EnableDeg),
    invalidate_lookup(ExplainSUBP,InvalidS,InvalidDeg),
    tolerate_lookup(ExplainSUBP,TolerateS,TolerateDeg),
    enable_explanation(ExplainSUBP,EnableS,EnableDeg),
    invalid_explanation(ExplainSUBP,InvalidS,InvalidDeg),
    tolerate_explanation(ExplainSUBP,TolerateS,TolerateDeg).

```

Two strings of text are associated with the description of the problem's enabling and invalidating conditions in the answer template. The difference between the two strings are that one is used if the problem has a single required symptom while the other is used when it has more than one required symptom.

```

/* -----> enabling condition template----- */
enable_explanation(ExplainSUBP,EnableS,EnableDeg):-
    length(EnableS,1),
    prompt(['The ',ExplainSUBP,' is deduced by checking your observed
symptoms for the required enabling symptom ',EnableS,'which is
required to match to the specified degree of ',EnableDeg,'.']).

enable_explanation(ExplainSUBP,EnableS,EnableDeg):-
    prompt(['The ',ExplainSUBP,' is deduced by matching your observed
symptoms against the required enabling symptoms, ']),
    write(EnableS),
    prompt(['to the specified degree of match of ',EnableDeg,'.']).

/* -----> invalid condition template----- */
invalid_explanation(ExplainSUBP,InvalidS,InvalidDeg):-
    length(InvalidS,1),
    prompt(['Once a ',ExplainSUBP,' is enabled to the specified degree, if its
invalidating symptom, ',InvalidS,' is observed, it is found to satisfy
the invalidating degree of match of ',InvalidDeg,'.']).

invalid_explanation(ExplainSUBP,InvalidS,InvalidDeg):-
    prompt(['Once a ',ExplainSUBP,' is enabled to the specified degree, its
observed invalidating symptoms, ']),
    write(InvalidS),
    prompt([' are checked against the invalidating degree of match specified to be
',InvalidDeg,'.']).

```

The tolerance condition template is generated in a similar manner to the enabling and invalidating conditions. However, it may have an extra symptom set description to explain. This explanation is generated when the problem to be explained does not have any tolerance symptoms. In such a case, the user is told that when the problem is found invalid, it has no special case symptoms to override the invalidation and thus, it cannot be deduced.

```

/* -----> tolerance condition template----- */
tolerate_explanation(ExplainSUBP,TolerateS,):-
    TolerateS = [none],
    prompt(['Since the ',ExplainSUBP,'has no special case symptoms, once it is
        found invalid, it cannot be deduced.']).

tolerate_explanation(ExplainSUBP,TolerateS,TolerateDeg):-
    length(TolerateS,1),
    prompt(['If the ',ExplainSUBP,'is found invalid, the observed symptoms are
        checked for its special case symptom, ',TolerateS,' specified for a
        degree of match of ',TolerateDeg,'. If the symptom is observed, it
        overrides the invalidation condition and ',ExplainSUBP,'is
        deduced.']).

tolerate_explanation(ExplainSUBP,TolerateS,TolerateDeg):-
    prompt(['If the ',ExplainSUBP,'is found invalid, the observed symptoms are
        checked against its special case symptoms, ']),
    write(TolerateS),ttflush,
    prompt([' for a specified degree of match of ',TolerateDeg,'.']),
    prompt(['Once a ',ExplainSUBP,'is enabled and found valid (by checking the
        special case symptoms if necessary), it is deduced.']).

```

Ability Explanation for Question 9

How do you deduce the alternate problem for the subproblem I'm specifying below? is another ability question that the user can ask. The template for this explanation is generated by the second clause of the *ability_explanation* predicate. The question is answered in a manner similar to that of the previous ability question. The name of the alternate problem, the required symptoms and degree of match are instantiated from the problem frame slot. One of three explanation templates is used depending on the constraints of the problem alternate. If there is no alternate, the user is told that the selected problem has no alternate. If the alternate problem can be triggered by only one symptom, a singular sentence is constructed to describe the symptom required. Conversely, if the alternate is triggered by several symptoms, a plural sentence describing the symptoms and the required degree of match is constructed.

```

/* -----ABILITY EXPLANATION TEMPLATE 9----- */
ability_explanation(QuestionN):-
    QuestionN = 9,
    prompted_line(['subproblem: ', ExplainSUBP_code),
    name(ExplainSUBP, ExplainSUBP_code),
    get_ability_Q(QuestionN, ExplainSUBP),
    alternate_lookup(ExplainSUBP, AlternateE, AlternateS, AlternateD),
    alt_explanation(ExplainSUBP, AlternateE, AlternateS, AlternateD).

```

```

alt_explanation(ExplainSUBP,AlternateE,_):-
    AlternateE = none,
    prompt(['The 'ExplainSUBP,' does not have an alternate, and as a result,
           none is deduced.']).

alt_explanation(ExplainSUBP,AlternateE,AlternateS,AlternateD):-
    length(AlternateS,1),
    prompt(['To deduce the alternate subproblem, 'AlternateE,', for a
           'ExplainSUBP,the alternate"s required symptom, 'AlternateS,' is']),
    rest_of_it(AlternateD).

alt_explanation(ExplainSUBP,AlternateE,AlternateS,AlternateD):-
    prompt(['To deduce the alternate subproblem, 'AlternateE,', for a
           'ExplainSUBP,the alternate"s required symptoms, '']),
    write(AlternateS),
    write(' are'),
    rest_of_it(AlternateD).

rest_of_it(AlternateD):-
    prompt(['checked against your observed symptoms. If a match of at least
           'AlternateD,'is achieved, then the alternate problem is deduced and
           refined as far as your observed symptoms allow.']).

```

Ability Explanation for Question 10

The third ability question the user can ask for is *How do you refine the subproblem I'm specifying below?* . Again, a similar approach to the generation of the previous ability explanations is used. The refinement subgoal set, the associated required symptoms and degrees of match are extracted from the problem frame. If the problem has no refinements, the user is told. Otherwise the recursive predicate *pretty_print_refs* is called to generate the strings of text to explain the name of the refinement goal and the required symptoms and degrees of match for its refinement. This predicate also calls a predicate to generate two strings of texts depending on whether the set of symptoms contains one or several symptoms.

```

/* -----ABILITY EXPLANATION TEMPLATE 10----- */
ability_explanation(QuestionN):-
    QuestionN = 10,
    prompted_line(['subproblem: ',ExplainSUBP_code),
    name(ExplainSUBP, ExplainSUBP_code),
    get_ability_Q(QuestionN,ExplainSUBP),
    refinement_lookup(ExplainSUBP,RefineE,RefineS,RefineD),
    ref_explanation(ExplainSUBP,RefineE,RefineS,RefineD).

ref_explanation(ExplainSUBP,RefineE,RefineS,RefineD):-
    RefineE \== no_refine_to_subG,
    prompt(['A ',ExplainSUBP,' can be refined as follows:']),
    pretty_print_refs(RefineE,RefineS,RefineD).

ref_explanation(ExplainSUBP,RefineE,_,_):-
    RefineE = no_refine_to_subG,
    prompt(['A ',ExplainSUBP,' does not have any refinement subgoals and as a
    result is not refined.']).

/* pretty print refinement subgoal and generate required refinement symptoms
and degrees -----> */
pretty_print_refs([],[],[]).
pretty_print_refs([R|Rs],[S|Ss],[D|Ds]):-
    explain_R(R,S,D),
    pretty_print_refs(Rs,Ss,Ds).

explain_R(R,S,D):-
    length(S,1),
    prompt(['--> to refine to subgoal ',R,' the refinement symptom, ',S]),
    prompt([' is to be observed with a degree of match of ',D,'.']).

explain_R(R,S,D):-
    prompt(['--> to refine to subgoal ',R,' the refinement symptoms, ']),
    write(S),
    prompt([' are to be observed with a degree of match of ',D,'.']).

```

4.5.2.4 Factual Explanations

The fourth type of explanations that a user can ask for are factual explanations. Note Appendix B5 for example explanations for RATIONALE's factual questions about the ACME domain.

Factual Explanation for Question 11a

Question 11a allows the user to ask, *What subproblems can the system handle?* . This is answered by the first clause of the explanation predicate *factual_explanation* . It calls the predicates *what_subGs_exist* to generate the names of the top level subgoals of the domain and *top_problem_explanation* to output the explanation to the user.

```
/* -----FACTUAL EXPLANATION TEMPLATE 11 a ----- */
factual_explanation(QuestionN):-
    QuestionN = '11a',
    get_factual_Q(QuestionN),
    what_subGs_exist(Domain,TopGNames),
    top_problem_explanation(Domain,TopGNames).

top_problem_explanation(Domain,Names):-
    prompt(['RATIOANLE running the 'Domain,' advisor, can handle the
           following top level problems:']),
    writeList(Names).
```


Factual Explanation for Question 11b

The second factual question the user can ask is question 11b, *What subproblems can the problem I'm specifying be focused to?* . This is answered by the predicate *subs_problem_explanation* after the predicate *what_subs_exist* is used to generate the problem refinement subgoals. Once the set of subgoals is returned, it is explained using one of three templates. The first is used if the user problem has no refinement subgoals. The next two templates are used to differentiate between the possibility of a single refinement and a multiple refinement for the chosen problem.

```

/* -----FACTUAL EXPLANATION TEMPLATE 11 b ----- */
factual_explanation(QuestionN):-
    QuestionN = '11b',
    get_problem(ExplainSUBP),
    get_factual_Q(QuestionN,ExplainSUBP),
    what_subs_exist(ExplainSUBP,SubNames),
    subs_problem_explanation(ExplainSUBP,SubNames).

subs_problem_explanation(ExplainSUBP,SubNames):-
    var(SubNames),
    prompt(['A ',ExplainSUBP,', has no subproblems to be focused further to.']).

subs_problem_explanation(ExplainSUBP,SubNames):-
    atom(SubNames),
    prompt(['A ',ExplainSUBP,', depending on symptoms, can be focused to a
    ',SubNames,']).

subs_problem_explanation(ExplainSUBP,SubNames):-
    prompt(['A ',ExplainSUBP,', depending on symptoms, can be focused to any
    one or all of the following subproblems:']),
    writeList(SubNames).

```

Factual Explanation for Questions 12a, 12b, 12c, 12d, 12e and 12f

The next six factual questions are related to the conditions required to satisfy the relevance, enablement, invalidation, tolerance, refinement and alternate requirements for a particular problem. All the questions are answered in a similar manner but use different strings of text for explanation. The form of question 12 is,

*What symptoms and respective degrees of match are
 12a the relevant 12b /the enabling
 12c /the invalidating 12d /the special case
 12e /the refinement 12f /the alternate
 symptoms and degrees of match for the subproblem I'm specifying below?*

The answer for question 12a is generated by finding the relevant symptom set for the selected problem and instantiating it into the explanation template for the question.

```
/* -----FACTUAL EXPLANATION TEMPLATE 12 a ----- */
factual_explanation(QuestionN):-
  QuestionN = '12a',
  get_problem(ExplainSUBP),
  get_factual_Q(QuestionN,ExplainSUBP),
  relevant_lookup(ExplainSUBP,RelevantS),
  relevant_symp_E(ExplainSUBP,RelevantS).

relevant_symp_E(ExplainSUBP,RelevantS):-
  prompt(['The symptoms relevant to a ',ExplainSUBP,' are:']),
  writeList(RelevantS).
```

The answers for questions 12b and 12c are generated in a similar manner. The enabling or invalidating symptoms are derived from the frame slot and the explanation is output using one of two possible strings. If the condition can be satisfied using only one symptom, a singular form of the explanation is generated, otherwise the plural form is used.

```
/* -----FACTUAL EXPLANATION TEMPLATE 12 b ----- */
```

```
factual_explanation(QuestionN):-
```

```
    QuestionN = '12b',
```

```
    get_problem(ExplainSUBP),
```

```
    get_factual_Q(QuestionN,ExplainSUBP),
```

```
    enable_lookup(ExplainSUBP,EnableS,EnableDeg),
```

```
    enable_symp_E(ExplainSUBP,EnableS,EnableDeg).
```

```
enable_symp_E(ExplainSUBP,EnableS,EnableDeg):-
```

```
    length(EnableS,1),
```

```
    prompt(['The symptom that enables a ',ExplainSUBP,' is a ',EnableS]),
```

```
    prompt([' and it must match to the degree of ',EnableDeg,'.']).
```

```
enable_symp_E(ExplainSUBP,EnableS,EnableDeg):-
```

```
    prompt(['The symptoms that enable a ',ExplainSUBP,' are:']),
```

```
    writeList(EnableS),nl,
```

```
    prompt(['and they must match to the degree of ',EnableDeg,'.']).
```

```
/* -----FACTUAL EXPLANATION TEMPLATE 12 c ----- */
```

```
factual_explanation(QuestionN):-
```

```
    QuestionN = '12c',
```

```
    get_problem(ExplainSUBP),
```

```
    get_factual_Q(QuestionN,ExplainSUBP),
```

```
    invalidate_lookup(ExplainSUBP,InvalidS,InvalidDeg),
```

```
    invalid_symp_E(ExplainSUBP,InvalidS,InvalidDeg).
```

```
invalid_symp_E(ExplainSUBP,InvalidS,InvalidDeg):-
```

```
    length(InvalidS,1),
```

```
    prompt(['The symptom that invalidates a ',ExplainSUBP,' is a ',InvalidS]),
```

```
    prompt([' and it must match to the degree of ',InvalidDeg,'.']).
```

```
invalid_symp_E(ExplainSUBP,InvalidS,InvalidDeg):-
```

```
    prompt(['The symptoms that invalidate a ',ExplainSUBP,' are:']),
```

```
    writeList(InvalidS),
```

```
    prompt(['and they must match to the degree of ',InvalidDeg,'.']).
```

Question 12d which allows the user to ask for an explanation of the tolerance or special case symptoms, has a third type of explanation associated with it. If the problem has no tolerance symptoms to override invalidation, an explanation is generated relating this to the user. Like the enabling and invalidating condition explanations, it also has the singular and plural string of text to explain a single or several symptom requirements.

```
/* -----FACTUAL EXPLANATION TEMPLATE 12 d ----- */
```

```
factual_explanation(QuestionN):-
```

```
    QuestionN = '12d',
```

```
    get_problem(ExplainSUBP),
```

```
    get_factual_Q(QuestionN,ExplainSUBP),
```

```
    tolerate_lookup(ExplainSUBP,TolerateS,TolerateDeg),
```

```
    tolerate_symp_E(ExplainSUBP,TolerateS,TolerateDeg).
```

```

tolerate_symp_E(ExplainSUBP,TolerateS):-
    TolerateS = [none],
    prompt(['The 'ExplainSUBP,' has no special case symptoms to override
            invalidation.']).

tolerate_symp_E(ExplainSUBP,TolerateS,TolerateDeg):-
    length(TolerateS,1),
    prompt(['The symptom that overrides the invalidation of the
            'ExplainSUBP,' is a 'TolerateS']),
    prompt([' and it must match to the degree of 'TolerateDeg,']).

tolerate_symp_E(ExplainSUBP,TolerateS,TolerateDeg):-
    prompt(['The symptoms that override the invalidation of the
            'ExplainSUBP,' are:']),
    writeList(TolerateS,nl,
    prompt(['and they must match to the degree of 'TolerateDeg,'])).

```

Questions 12e and 12f allow the user to ask for an explanation of the refinement and alternate conditions of a problem. Their explanations are generated in a very similar manner. Three strings of text explain the requirements associated with refinement or generating alternates. If the problem has no refinement subgoals or no alternate problem, the user is told. If it has a single symptom for satisfying the condition in question, a string of text in singular form is output to the user. Otherwise, a plural form of the string of text explaining the symptom requirements is output of the user.

```

/* -----FACTUAL EXPLANATION TEMPLATE 12 e ----- */
factual_explanation(QuestionN):-
    QuestionN = '12e',
    get_problem(ExplainSUBP),
    get_factual_Q(QuestionN,ExplainSUBP),
    refinement_lookup(ExplainSUBP,RefineE,RefineS,RefineDeg),
    refinement_symp_E(ExplainSUBP,RefineE,RefineS,RefineDeg).

refinement_symp_E(ExplainSUBP,RefineE,_,_):-
    RefineE = no_refine_to_subG,
    prompt(['The 'ExplainSUBP,' has no refinement subgoals.']).

refinement_symp_E(ExplainSUBP,RefineE,RefineS,RefineDeg):-
    atom(RefineE),
    length(RefineS,1),
    prompt(['The symptom that triggers the refinement subproblem, 'RefineE,'
            for the 'ExplainSUBP,' is a 'RefineS']),
    prompt([' and it must match to the degree of 'RefineDeg,'])).

```

```

refinement_symp_E(ExplainSUBP,RefineE,RefineS,RefineDeg):-
    atom(RefineE),
    prompt(['The symptoms that trigger the refinement subproblem, 'RefineE,'
           for the 'ExplainSUBP,' are:']),
    writeList(RefineS),nl,
    prompt(['and they must match to the degree of 'RefineDeg,']).

refinement_symp_E(ExplainSUBP,RefineE,RefineS,RefineDeg):-
    prompt(['Several refinements are possible for a 'ExplainSUBP,']),
    pretty_print_refs(RefineE,RefineS,RefineDeg).

/* -----FACTUAL EXPLANATION TEMPLATE 12 f -----
 */
factual_explanation(QuestionN):-
    QuestionN = '12f',
    get_problem(ExplainSUBP),
    get_factual_Q(QuestionN,ExplainSUBP),
    alternate_lookup(ExplainSUBP,AlternateE,AlternateS,AlternateDeg),
    alternate_symp_E(ExplainSUBP,AlternateE,AlternateS,AlternateDeg).

alternate_symp_E(ExplainSUBP,AlternateE,_):-
    AlternateE = none,
    prompt(['The 'ExplainSUBP,' has no alternate problem.']).

alternate_symp_E(ExplainSUBP,AlternateE,AlternateS,AlternateDeg):-
    length(AlternateS,1),
    prompt(['The symptom that triggers the alternate subproblem,
           'AlternateE,' of the 'ExplainSUBP,' is a 'AlternateS']),
    prompt(['and it must match to the degree of 'AlternateDeg,']).

alternate_symp_E(ExplainSUBP,AlternateE,AlternateS,AlternateDeg):-
    atom(AlternateE),
    prompt(['The symptoms that trigger the alternate subproblem, 'AlternateE,'
           of the 'ExplainSUBP,' are:']),
    writeList(AlternateS),nl,
    prompt(['and they must match to the degree of 'AlternateDeg,']).

alternate_symp_E(ExplainSUBP,AlternateE,AlternateS,AlternateDeg):-
    prompt(['Several alternate refinements are possible for a 'ExplainSUBP,']),
    pretty_print_refs(AlternateE,AlternateS,AlternateDeg).

```

Chapter 5

Summary

5.1 Concluding Remarks

Most existing explanation facilities are grafted onto an independently designed inference engine. RATIONALE departs from this approach. Even in its current, preliminary implementation, it behaves as though it had to deliberate about and to explain each refinement step to itself. By treating explanation as primary, RATIONALE forces the system designer to represent knowledge explicitly that might otherwise be left implicit. As is the case with explanation facilities in XPLAIN, EES and MDX, RATIONALE's knowledge organization and representation serve to provide the designer with a powerful hypothesis-based modeling scheme which simplifies the overall task of domain modeling and knowledge acquisition.

The knowledge to be rendered explicit includes knowledge as to why a particular rule or hypothesis is preferred in a given situation to a reasonably looking alternative, why certain exceptions are acceptable in some situations but not in others, and why certain global inference strategies succeed in certain circumstances but fail in others. Directing the

reasoning strategies in such an explicit manner simplifies the task of explanation thus making explicit reasoning and explanation equivalent.

Categorizing user questions into event, hypothetical, ability and factual questions is an important contribution of this work. The categories provide a path through the reasoning strategies for the user to follow. Such a path is not as directly obvious in other well-structured systems such as XPLAIN and MDX. Several extensions to RATIONALE are possible. They are described in sections 5.2.1 to 5.2.5 below.

5.2 Discussion and Extensions to RATIONALE

5.2.1 Modeling Domain Knowledge

Modeling the knowledge of the domain using a front end knowledge acquisition tool is one extension to RATIONALE. This extension would include the domain definition and editing modules described in Chapter 3, the design chapter. The tool would also include a knowledge base *browser*. The browser would be used to graphically represent the objects in the knowledge base according to their relation to other objects. The graphed hierarchy would consist of more than the inheritance hierarchies of *is-a* and *part-of* links. It would also graph shared relations between objects.

5.2.2 Learning by Generating Intermediate Hypotheses

Learning as a source of new domain knowledge is another extension. RATIONALE reasons and generates an explanation even if the top level hypothesis could not be focused into any of its siblings or to an alternate hypothesis. In some cases RATIONALE fails to focus a problem because the observed symptoms do not fully satisfy the required degree of match for any of the problem's refinements. Identifying the failure to refine a problem is already tracked for the purpose of explanation. A mechanism to suggest an intermediate hypothesis for this failed refinement is what the system requires to learn. Such a mechanism is not a trivial one. It would have to identify the symptoms and suggest to the domain expert a manner by which the symptoms can diagnose a hypothesis which fits the domain model. Included in this fit would be the suggested breakdown of the symptoms into enabling, invalidating, tolerance, refinement and alternate problem symptom sets. In addition, the mechanism could suggest a string of text to explain the hypothesis. Generating the string adds another level of complexity to the mechanism, but is necessary to complete the generation of an intermediate hypothesis.

5.2.3 User Modeling in Explanation

RATIONALE could take into account user modeling information. Information such as the user selections, questions about the reasoning strategies and errors could be tracked to form a simple model of the user. Such tracking allows RATIONALE to construct explanations that are better directed at the level of knowledge of the user. The explanations

could then be generated to expand the knowledge of the user according to domain generalization or specialization hierarchies.

5.2.4 Explaining Interactions of Multiple Domains

An area that has to be addressed further in RATIONALE is that of multiple domains. Assuming that several domain hierarchies are defined using the object definition utility, RATIONALE can be used to diagnose a problem in any of the defined domains. The top goal object definition would then be the key to the selection of the correct hierarchy. The knowledge of the different hierarchies has to be modeled well enough to permit the selection of one domain hierarchy over another. In addition, reasoning about alternate problems could be extended to reasoning about alternate domains. To bound the problem of searching through multiple alternate domains, the represented knowledge has to be explicit and focused enough to allow bounded traversal through the domain hierarchies.

5.2.5 Explaining Evidential Reasoning

RATIONALE currently refines a problem based on the user's selection of a problem or of problem-enabling symptom pairs. Symptoms are all weighted equally. An extension to RATIONALE's reasoning algorithm would be to weight symptoms according to their contribution to the correct diagnosis of a hypothesis. Thus, symptoms that affect a

diagnosis more than others should be given a higher weight. This weighting should be included in the explanations derived from RATIONALE's explicit reasoning mechanism.

In addition, RATIONALE should allow the user to input observed symptoms without biasing them with the problem they diagnose. Thus, a user who selects unknown as the problem should be given a list of symptoms rather than a list of problem-enabling symptom pairs. RATIONALE could then infer the problem based on the selected enabling symptoms.

References

[Aikins83]

Aikins J.S. Prototypical Knowledge for Expert Systems. *Artificial Intelligence*, Vol. 20, 163-210 (1983).

[Bobrow86]

Bobrow D.G., Mittal S. and Stefik M.J. Expert Systems: Perils and Promise. *Communications of the ACM*, Vol. 29, No. 9, 880-894 (September 1986).

[Brady83]

Brady M. and Berwick R.C. Computational Models Of Discourse. MIT Press, Massachusetts (1983).

[Buchanan84]

Buchanan B.G. and Shortliffe E.H. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley Publishing Company, Massachusetts (1984).

[Bylander86]

Bylander T. and Mittal S. CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling. *The AI Magazine*, 66-77 (August 1986).

[Chandrasekaran79]

Chandrasekaran B., Gomez F., Mittal S. and Smith J.W. An Approach to Medical Diagnosis Based on Conceptual Structures. *Proceedings of the Joint Conference on Artificial Intelligence*, 134-142 (1979).

[Chandrasekaran82]

Chandrasekaran B., Mittal S. and Smith J.W. Reasoning With Uncertain Knowledge: The MDX Approach. *Proceedings of 1st Annual Joint Conf. American Informatics Assoc.*, 335-339 (May 1982).

[Chandrasekaran83]

Chandrasekaran B. and Mittal S. Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-Solving. International Journal of Man-Machine Studies, Vol. 19, No. 5, 425-436 (November 1983).

[Chandrasekaran86a]

Chandrasekaran B. Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. IEEE Expert, Vol. 1, No. 3, 23-30 (1986).

[Chandrasekaran86b]

Chandrasekaran B., Josephson J. and Keuneke A. Functional Representations as a Basis for Generating Explanations. IEEE International Conference Proceedings on Systems, Man and Cybernetics 726-731 (1986).

[Chandrasekaran86c]

Chandrasekaran B. and Tanner M.C. Uncertainty Handling in Expert Systems: Uniform vs. Task-Specific Formalisms. Uncertainty in Artificial Intelligence, Kanal L.N. and Lemmer J., editors, North Holland Publishing (1986).

[Clancey83]

Clancey W.J. The Epistemology of a Rule-Based Expert System - A Framework for Explanation. Artificial Intelligence 20 (3), 215-251 (1983).

[Clancey86]

Clancey W.J. From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985. The AI Magazine, 40-60 (August 1986).

[Clancey87]

Clancey W.J. Knowledge-Based Tutoring: The GUIDON Program, MIT Press, Cambridge, Massachusetts (1987).

[Clocksin84]

Clocksin W.F. and Mellish C.S. Programming in Prolog, 2nd edition, Springer-Verlag, New York (1984).

[Dudzinski86]

Dudzinski E.C., Brink J.R. and Sharma D.D. CSRL - From Laboratory to Industry. IEEE Proceedings of Expert Systems in Government Symposium. 266-272 (22-24 October 1986).

[Ganascia84]

Ganascia J.G. Explanation Facilities for Diagnosis Systems. Cybernetics And Systems Research 2. Proceedings of the 7th European Meeting., 805-810 (24-27 April 1984).

[Goguen83]

Goguen J.A., Weiner J.L. and Linde C. Reasoning And Natural Explanation. International Journal Of Man-Machine Studies, Vol. 19, No. 6, 521-559 (Dec 1983).

[Gomez81]

Gomez F. and Chandrasekaran B. Knowledge Organization and Distribution for Medical Diagnosis. IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No. 1, 34-43 (Jan. 1981).

[Hardman85]

Hardman L. Expert User-Friendly, Systems International (GB) Vol. 13, No. 75-76, (April 1985).

[Hayes82]

Hayes P.J. and Glasner I.D. Automatic Construction of Explanation Networks for a Cooperative User Interface. SIGSOC Bulletin (USA) Vol. 13, No. 2-3, 6-14, (Jan. 1982).

[Hayes83]

Hayes P.J. and Reddy D.R. Steps Towards Graceful Interaction in Spoken and Written Man-Machine Communication. International Journal of Man-Machine Studies 19, No. 3, 231-284 (Sept. 1983).

[Hasling84]

Hasling D.W., Clancey W.J. and Rennels G. Strategic Explanations for a Diagnostic Consultation System. International Journal of Man-Machine Studies Vol. 20, 3-19 (1984).

[Josephson84]

Josephson J.R., Chandrasekaran B. and Smith J. W. Assembling the Best Explanation. Proceedings IEEE Workshop on Principles of Knowledge-Based Systems, IEEE Computer Society, Calif., 185-190 (Dec. 1984).

[Mittal84]

Mittal S., Chandrasekaran B. and Sticklen J. Patrec: A Knowledge-Directed Database for a Diagnostic Expert System. IEEE Computer, Vol. 17, No. 9, 51-58 (September 1984).

[Neches85]

Neches R., Swartout W.R., and Moore J.D. Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development. IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, 1337-1351 (November 1985).

[Punch86]

Punch III W.F., Tanner M.C., and Josephson J. Design Considerations for PIERCE, a High-Level Language for Hypothesis Assembly. IEEE Conference Proceedings of Expert Systems in Government Symposium. 279-281(22-24 October 1986).

[Ramsey86]

Ramsey C.L., Reggia J.A., Nau D.S., and Ferrentino A. A Comparative Analysis of Methods for Expert Systems. International Journal Man-Machine Studies, 24, 5, 475-499 (May 1986).

[Scott77]

Scott A.C., Clancey W.J., Davis R. and Shortliffe E.H. Explanation Capabilities of Production-Based Consultation Systems. Stanford University, Dept. of Computer Science, Report No. STAN-CS-77-593, 35p (Feb. 1977).

[Stabler86]

Stabler E.P., Object-Oriented Programming in Prolog, AI Expert, Oct. 1986, 46-57.

[Steels85]

Steels L. Second Generation expert systems. Future Generation Computer Systems, 213-221 (June 1985).

[Sterling86]

Sterling L. and Shapiro E. The Art of Prolog. MIT Press, Cambridge, Massachusetts (1986).

[Swartout81]

Swartout W.R Producing Improved Explanations and Justifications of Expert Consulting Programs Using an Automatic Programming Approach. PhD Thesis, MIT, (1981).

[Swartout82]

Swartout W.R. and Blazer R. On the Inevitable Intertwining of Specification and Implementation. Communications ACM Vol. 25, No. 7, 438-439 (July 1982).

[Swartout83]

Swartout W.R. XPLAIN: A System for Creating and Explaining Consulting Programs. Artificial Intelligence Vol. 21, No. 3, 285-325 (Sept. 1983).

[Swartout87]

Swartout W.R. and Smoliar S.W. On Making Expert Systems More Like Experts. Expert Systems Vol. 4, No. 3, 196-207 (August 1987).

[Wallis82]

Wallis J.W. and Shortliffe E.H. Explanatory Power for Medical Expert Systems: Studies in the Representation of Causal Relationships for Clinical Consultations. Meth. Inform. Med. (Germany) Vol. 21, No. 3, 127-136 (July 1982).

[Weiner80a]

Weiner J.L. BLAH, A System Which Explains its Reasoning. Artificial Intelligence Vol. 15, No. 1,2, 19-48 (November 1980).

[Weiner80b]

Weiner J.L. and Burstall R.M. Making Programs More Readable. Proceedings of the Fourth International Symposium on Programming. 22-24 April, 1980, Paris, France. 327-341(1980).

Appendices

Appendix A: Object Definition of ACME Workstation

Note: the construction of the ACME hierarchy is described in Chapter 4 under the description of RATIONALE's Frame Generator in sections 4.2.1 which describes the top goal definition and section 4.2.2 which describes the subgoal definition.

/* This is a building of the ACME knowledge hierarchy by calling Stabler's object facility .

first----> define add_object for a generalized top goal

----> instantiate with the top goal of expert system (e.g. MAD, FIXIT, etc.)

second---> define add_object for a generalized top level sub goal

---> instantiate with an instance of each top level sub goal

third----> instantiate the generalized sub goal with the refined sub goal
of each of the top level sub goals

(NB: the code for this is identical to the code of the top level subgoals)

This is the format of the add_object call to instantiate the goals, note that there are 2 types:

1) top_goal object:

```
add_topgoal(TG_name,  
            AIM,  
            problem_type(EC,IC,TC)).
```

2) sub_goal object:

```
add_subgoal(SG_name_refined_from,Ref_SG_name,  
            explanation(NoviceE,ExperiencedE,ExpertE),  
            explanation_detail(SubG_Concept_Complexity,Detail_Importance),  
            enabling_cond(Enabling_Symp,EDegree),  
            invalidate_cond(Invalid_Symp,IDegree),  
            tolerate_cond(Special_Case_Symp,TDegree),  
            alternate_subG(Alternate_SG_name,  
                           Shared_Symp,  
                           ADegree),  
            Urgency,  
            refinement_subG(Refine_to_SG_name,Suff_Set,RDegree)).  */
```

```

:-
add_topgoal(acme,
    fix_workstation,
    'fix ACME workstation',
    problem_type(workstation,server,network)).

:-
add_subgoal(fix_workstation,
    software_problem,
    explanation('There may be a problem with ACME workstation software,
        run the diagnostics.',
        'Workstation software problem, check diagnostics.',
        'Workstation software bug or corruption, run tests.'),
    explanation_detail(3,10),
    enabling_cond([failed_boot,memory_error,software_process_lost],
        0.3),
    invalidate_cond([hardware_error],1.0),
    tolerate_cond([failed_boot,processor_error,memory_error],0.3),
    alternate_subG(hardware_problem,
        [memory_error,software_process_lost],
        0.5),
    3,
    refinement_subG([disk_crash,
        processor_software_problem,
        memory_problem],
        [[hard_disk_access_denied],
        [failed_boot,memory_error],[memory_error]],
        [1.0,0.5,1.0])).

:-
add_subgoal(software_problem,
    disk_crash,
    explanation('There is a problem with workstation hard disk software and it
        may require reloading. Check hard disk diagnostics.',
        'Hard disk software problem, run disk diagnostics.',
        'Hard disk software fault needs tracing.'),
    explanation_detail(5,4),
    enabling_cond([software_process_lost,hard_disk_access_denied],
        0.5),
    invalidate_cond([power_error],1.0),
    tolerate_cond([connector_problem],1.0),
    alternate_subG(processor_card_problem,
        [keyboard_locked,screen_blank,mouse_failing],
        1.0),
    3,
    refinement_subG(disk_sector_problem,
        [hard_disk_access_denied],1.0)).

```

3 of/de 3

MicroD

```

:-
add_subgoal(disk_crash,
            disk_sector_problem,
            explanation('There is a problem with a sector of the hard disk of the
disk sector diagnostics and contact ACME service
                        department.',
                        'Hard disk sector problem, run disk diagnostics and call for
                        service.',
                        'Disk sector fault needs tracing, call service with results.'),
            explanation_detail(8,7),
            enabling_cond([software_process_lost,hard_disk_access_denied],
                          0.5),
            invalidate_cond([power_error],1.0),
            tolerate_cond([connector_problem],1.0),
            alternate_subG(none,
                          [none],
                          0.0),
            3,
            refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(software_problem,
            processor_software_problem,
            explanation('There may be a problem with workstation micro-processor
                        software and it may require reloading. Check
                        processor diagnostics.',
                        'Processor software problem, run processor diagnostics.',
                        'Processor software bug or corruption needs tracing.'),
            explanation_detail(5,5),
            enabling_cond([failed_boot,memory_error,processor_error],
                          0.3),
            invalidate_cond([power_error],1.0),
            tolerate_cond([connector_problem],1.0),
            alternate_subG(processor_card_problem,
                          [processor_error],
                          1.0),
            2,
            refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(software_problem,
            memory_problem,
            explanation('There may be a problem with workstation memory software
                        and it may require reloading. Run memory diagnostics.',
                        'Memory software problem may need reloading.',
                        'Memory software bug or corruption requires reload.'),
            explanation_detail(5,8),
            enabling_cond([memory_error,memory_access_denied],
                        0.5),
            invalidate_cond([power_error],1.0),
            tolerate_cond([none],0.0),
            alternate_subG(memory_card_problem,
                        [memory_error,memory_access_denied],
                        1.0),
            2,
            refinement_subG([memory_segment_problem,
                        memory_partition_problem],
                        [[memory_access_denied],[failed_boot]],
                        [1.0,1.0])).

```

```

:-
add_subgoal(memory_problem,
            memory_segment_problem,
            explanation('There may be a problem with a segment of workstation
                        memory. Run memory diagnostics and contact ACME
                        service.',
                        'Memory segment problem may need ACME service, try
                        diagnostics.',
                        'Bad memory segment, trace and call for service'),
            explanation_detail(8,5),
            enabling_cond([memory_error,memory_access_denied],
                        0.5),
            invalidate_cond([power_error],1.0),
            tolerate_cond([none],0.0),
            alternate_subG(memory_bank_problem,
                        [memory_error,memory_access_denied],
                        1.0),
            2,
            refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(memory_problem,
             memory_partition_problem,
             explanation('There may be a problem with the memory partition. Try re-
                           loading a new partition to work in.',
                           'Memory partition corrupted, reload clean one.',
                           'Partition corrupted, reload.'),
             explanation_detail(6,5),
             enabling_cond([failed_boot],
                           1.0),
             invalidate_cond([power_error],1.0),
             tolerate_cond([none],0.0),
             alternate_subG(memory_bank_problem,
                             [memory_error,memory_access_denied],
                             1.0),
             1,
             refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(fix_workstation,
             hardware_problem,
             explanation('There may be a problem with ACME workstation hardware.
                           Run hardware diagnostics.',
                           'Workstation hardware problem. Run hardware diagnostics.',
                           'Workstation hardware failure needs tracing.'),
             explanation_detail(3,10),
             enabling_cond([keyboard_locked,screen_blank,mouse_failing,
                           memory_error,software_process_lost],
                           0.5),
             invalidate_cond([power_error],1.0),
             tolerate_cond([none],0.0),
             alternate_subG(software_problem,
                             [memory_error,software_process_lost],
                             0.5),
             3,
             refinement_subG([processor_card_problem,memory_card_problem],
                             [[keyboard_locked,screen_blank,mouse_failing],
                              [software_process_lost,memory_error]],
                             [1.0,0.5])).

```



```

:-
add_subgoal(hardware_problem,
    processor_card_problem,
    explanation('There may be a problem with workstation micro-processor
        card and it may require replacing. Run card diagnostics and
        replace if it fails.',
        'Processor card problem. May need swapping.',
        'Processor card failure. Trace and swap.'),
    explanation_detail(5,7),
    enabling_cond([keyboard_locked,
        screen_blank,mouse_failing,
        processor_error],
        0.75),
    invalidate_cond([power_error],1.0),
    tolerate_cond([none],0.0),
    alternate_subG([disk_crash,processor_software_problem],
        [[hard_disk_access_denied],[processor_error]],
        {1.0,1.0}),
    3,
    refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(hardware_problem,
    memory_card_problem,
    explanation('There may be a problem with workstation memory card and it
        may require replacing. Run card diagnostics and replace it if
        it fails.',
        'Memory card problem. May need swapping.',
        'Memory card failure. Trace and swap.'),
    explanation_detail(6,6),
    enabling_cond([software_process_lost,memory_error,
        memory_access_denied],
        0.6),
    invalidate_cond([power_error],1.0),
    tolerate_cond([none],0.0),
    alternate_subG(memory_problem,
        [software_process_lost,memory_error],
        1.0),
    2,
    refinement_subG(memory_bank_problem,
        [memory_error,memory_access_denied],
        0.5)).

```

```

:-
add_subgoal(memory_card_problem,
             memory_bank_problem,
             explanation('There may be a problem with a bank of chips on the
                           workstation memory card and they may require replacing. Run
                           memory bank diagnostics and call for service if it fails tests.',
                           'Memory bank problem on card. May need to swap out bad
                           chips, test and call service',
                           'Memory bank failure. Trace and call service to swap.'),
             explanation_detail(8,7),
             enabling_cond([memory_error,memory_access_denied],
                           0.5),
             invalidate_cond([power_error],1.0),
             tolerate_cond([none],0.0),
             alternate_subG([memory_segmemt_problem,memory_partition_problem],
                           [[memory_error,memory_access_denied],
                           [failed_boot]],
                           [1.0,1.0]),
             2,
             refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(fix_workstation,
             communication_problem,
             explanation('There may be a problem with ACME communication with the
                           network. Run communication diagnostics.',
                           'Workstation communication problem. Run diagnostics.',
                           'Workstation not communicating, trace failure.'),
             explanation_detail(3,10),
             enabling_cond([failed_server_access],
                           1.0),
             invalidate_cond([software_error,hardware_error],0.5),
             tolerate_cond([connector_error],0.5),
             alternate_subG(memory_problem,
                           [memory_access_denied],
                           1.0),
             2,
             refinement_subG(server_problem,
                           [failed_server_access],
                           1.0)).

```

```

:-
add_subgoal(communication_problem,
            server_problem,
            explanation("The problem with workstation-server communication may be
                        due to a server problem. Run server diagnostics or contact
                        administrator.",
                        'Check file server since it may be causing the communication
                        failures.',
                        'Check server for communication fault.'),
            explanation_detail(7,5),
            enabling_cond([failed_server_access],
                        1.0),
            invalidate_cond([hardware_error,power_error],0.5),
            tolerate_cond([software_error],1.0),
            alternate_subG(none,
                        [none],
                        0.0),
            2,
            refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

:-
add_subgoal(server_problem,
            ethernet_problem,
            explanation("The problem with workstation-server communication may be
                        due to a server-ethernet communication problem. Contact
                        administrator.",
                        'Check ethernet-file server connection since it may be causing
                        the communication failures. Run ethernet-server
                        diagnostics.',
                        'Trace ethernet-server communication problem.'),
            explanation_detail(8,5),
            enabling_cond([failed_server_access],
                        1.0),
            invalidate_cond([power_error],1.0),
            tolerate_cond([none],1.0),
            alternate_subG(none,
                        [none],
                        0.0),
            2,
            refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

```

add_subgoal(fix_workstation,
    power_problem,
    explanation("There may be a problem with ACME workstation power.
        Run power diagnostics and call technician.",
        'Workstation power problem. Run diagnostics and contact
        technician.',
        'Workstation power fault. Trace.'),
    explanation_detail(3,10),
    enabling_cond([fuse_error,on_off_switch_ineffective],
        0.5),
    invalidate_cond([software_error,hardware_error],1.0),
    tolerate_cond([none],0.0),
    alternate_subG(none,
        [none],
        0.0),
    2,
    refinement_subG(no_refine_to_subG,[no_suff_set],0.0)).

```

Appendix B: RATIONALE Explanation Examples

How to Read Appendices B1 to B5

Appendices B1 to B5 contain test sessions of RATIONALE running on the Sun 3/60. The sessions follow this convention:

- variables substituted into the template appear as ***bold italics***
- selections that the user types in or copies in from the prompt menu appear as **bold**
- a line of ~~~~~ symbols separate the results of a new session or a user action from the main menu
- a line of ----- and ~~~~~ separate new sessions

B1: Main Menu Options

Quintus Prolog Release 2.2 (Sun-3, Unix 3.5)

Copyright (C) 1987, Quintus Computer Systems, Inc. All rights reserved.

1310 Villa Street, Mountain View, California (415) 965-7700

| ?-[demo6].

[consulting /usr/nrcsun4/suhayya/rat/prolog/demo6.pl...]

[compiling /usr/nrcsun4/suhayya/rat/prolog/objectutility.pl...]

[objectutility.pl compiled 1.633 sec 1,896 bytes]

[compiling /usr/nrcsun4/suhayya/rat/prolog/objclasses.pl...]

[objclasses.pl compiled 1.300 sec 1,332 bytes]

[compiling /usr/nrcsun4/suhayya/rat/prolog/acmeobj1.pl...]

[acmeobj1.pl compiled 3.400 sec 35,136 bytes]

[compiling /usr/nrcsun4/suhayya/rat/prolog/reasoner3.pl...]

[compiling /usr/prolog/q2.2/library/basics.pl...]

[basics.pl compiled in module basics 0.984 sec 1,472 bytes]

[compiling /usr/prolog/q2.2/library/readin.pl...]

[readin.pl compiled in module read_in 1.833 sec 1,676 bytes]

[compiling /usr/prolog/q2.2/library/sets.pl...]

[sets.pl compiled in module sets 4.834 sec 6,284 bytes]

[compiling /usr/prolog/q2.2/library/prompt.pl...]

[compiling /usr/prolog/q2.2/library/ctypes.pl...]

[compiling /usr/prolog/q2.2/library/between.pl...]

[compiling /usr/prolog/q2.2/library/types.pl...]

[types.pl compiled in module types 4.300 sec 5,564 bytes]

[between.pl compiled in module between 7.283 sec 8,264 bytes]

[ctypes.pl compiled in module ctypes 10.850 sec 13,920 bytes]

[compiling /usr/prolog/q2.2/library/lineio.pl...]

[lineio.pl compiled in module lineio 0.967 sec 1,528 bytes]

[prompt.pl compiled in module prompt 13.433 sec 17,660 bytes]

[compiling /usr/prolog/q2.2/library/lists.pl...]

[lists.pl compiled in module lists 7.300 sec 8,832 bytes]

[compiling /usr/prolog/q2.2/library/queues.pl...]

[queues.pl compiled in module queues 3.033 sec 3,976 bytes]

[compiling /usr/prolog/q2.2/library/flatten.pl...]

[flatten.pl compiled in module flatten 2.934 sec 3,724 bytes]

[reasoner3.pl compiled 44.283 sec 54,696 bytes]

[compiling /usr/nrcsun4/suhayya/rat/prolog/explainer4.pl...]

[explainer4.pl compiled 31.650 sec 34,584 bytes]

[demo6.pl consulted 82.716 sec 128,628 bytes]

yes

| ?-

~~~~~  
Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice  
experienced  
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: l: novice  
Your user level has been set to novice.

What type of problem have you been having with *acme* ?

software\_problem  
hardware\_problem  
communication\_problem  
power\_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: l: communication\_problem

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *communication\_problem*

software\_error  
hardware\_error  
connector\_error  
memory\_access\_denied  
failed\_server\_access

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of communication\_problem:

l: connector\_error

l: failed\_server\_access.

Your *communication\_problem* can be FOCUSED to give the following explanations:

The *server\_problem* indicates that:

The problem with workstation-server communication may be due to a *server-ethernet communication problem*. Contact administrator.

~~~~~

What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for communication_problem
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: l: 2

To *re-select top level subproblem*, please reply to the following.

What type of problem have you been having with *acme* ?

software_problem

hardware_problem

communication_problem

power_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: l: software_problem

To *re-select subproblem symptoms*, please reply to the following.

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

- If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *software_problem*

hardware_error
processor_error
software_process_lost
hard_disk_access_denied
failed_boot
memory_error

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of *software_problem*:

!processor_error
!software_process_lost
!hard_disk_access_denied
!failed_boot.

Your *software_problem* can be FOCUSED to give the following explanations:

The *disk_crash* indicates that:

There is a problem with a *sector of the hard disk* of the workstation and it may require a hard disk overhaul. Run hard disk sector diagnostics and contact ACME service department.

The *processor_software_problem* indicates that:

There may be a problem with workstation *micro-processor software* and it may require reloading. Check processor diagnostics.

Your *software_problem* has an *alternate* - a *hardware_problem*.

The *hardware_problem* indicates that:

There may be a problem with *workstation memory card* and it may require replacing. Run card diagnostics and replace it if it fails.

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for communication\_problem
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: 1: 3.

To *re-select subproblem symptoms*, please reply to the following.

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *communication\_problem*

software\_error  
hardware\_error  
connector\_error  
memory\_access\_denied  
failed\_server\_access

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of communication\_problem:

1: software\_error  
1: hardware\_error  
1: connector\_error  
1: memory\_access\_denied  
1: failed\_server\_access.

Your *communication\_problem* can be FOCUSED to give the following explanations:

The *server\_problem* indicates that:

The problem with workstation-server communication may be due to a *server-ethernet communication problem*. Contact administrator.

Your *communication\_problem* has an *alternate* - a *memory\_problem*.

The *memory\_problem* indicates that:

There may be a problem with a *segment of workstation memory*. Run memory diagnostics and contact ACME service.

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
2. re-select problem
3. re-select symptoms for *communication_problem*
4. ask questions about session results
5. quit RATIONALE running acme

user wants to: 1: 5

Thank you for using RATIONALE running the acme workstation problem advisor.
~~~~~

B2: Event Question Explanations

Note that an expert user familiar with the domain hierarchy can input a subgoal at a lower level than the top level subgoals. This is possible since RATIONALE checks that the problem name is a valid one in the hierarchy rather than only a valid top level subgoal. This is described in the thesis in section 4.3.1 under the heading 'State 12 - Select a User Problem'.

=====

Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice
experienced
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: l: expert

Your user level has been set to expert.

What type of problem have you been having with *acme* ?

software_problem
hardware_problem
communication_problem
power_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: l: disk_crash

RATIONALE is considering *RELEVANT SYMPTOMS* that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the *RELEVANT* symptoms listed below have all been observed, please type all.

If not all the *RELEVANT* listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of disk_crash

software_process_lost
power_error
connector_problem
keyboard_locked
screen_blank
mouse_failing
hard_disk_access_denied

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of disk_crash :

!software_process_lost
!connector_problem
!keyboard_locked
!screen_blank
!mouse_failing
!hard_disk_access_denied.

Your *disk_crash* can be FOCUSED to give the following explanations:

The *disk_sector_problem* indicates that:

Disk sector fault needs tracing, call service with results.

Your *disk_crash* has an alternate - a *processor_card_problem*.

The *processor_card_problem* indicates that:

Processor card failure. Trace and swap.

~~~~~

What would you like to do next?

1. re-select user level and re-display results
2. re-select problem
3. re-select symptoms for disk_crash
4. ask questions about session results
5. quit RATIONALE running acme

user wants to: l: 2

To *re-select top level subproblem*, please reply to the following.

What type of problem have you been having with *acme* ?

software_problem
hardware_problem
communication_problem
power_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: l: memory_problem

To *re-select subproblem symptoms*, please reply to the following.

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *memory_problem*

power_error
memory_error
memory_access_denied
failed_boot

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of memory_problem:

l:memory_error
l:memory_access_denied
l:failed_boot.

Your *memory_problem* can be FOCUSED to give the following explanations:

The *memory_segment_problem* indicates that:
Bad memory segment, trace and call for service

The *memory_partition_problem* indicates that:
Partition corrupted, reload.

Your *memory_problem* has an *alternate* - a *memory_card_problem*.

The *memory_card_problem* indicates that:
Memory bank failure. Trace and call service to swap.

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for disk\_crash
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: l: 3

To *re-select subproblem-symptoms*, please reply to the following.  
RATIONALE is considering RELEVANT SYMPTOMS that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *disk\_crash*

software\_process\_lost  
 power\_error  
 connector\_problem  
 keyboard\_locked  
 screen\_blank  
 mouse\_failing  
 hard\_disk\_access\_denied

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of *disk\_crash*:

l:software\_process\_lost  
 l:connector\_problem  
 l:keyboard\_locked  
 l:screen\_blank  
 l:mouse\_failing.

Your *disk\_crash* can be FOCUSED to give the following explanations:

Your *disk\_crash* has an *alternate* - a *processor\_card\_problem*.

The *processor\_card\_problem* indicates that:  
 Processor card failure. Trace and swap.

~~~~~

What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for *disk_crash*
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: l: 4

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *disk_crash*, WHY do you need the symptoms I'm specifying below?
- 2 For my *disk_crash*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *disk_crash*, WHY did you conclude the *disk_crash* or the subproblem I'm specifying below?
- 4 For my *disk_crash*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my *disk_crash*, HOW did you deduce the *disk_crash* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *disk_crash* ?
- 7 What-if the subproblem I'm specifying replaced the *disk_crash* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
 - 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are
 - 12a the relevant 12b /the enabling
 - 12c /the invalidating 12d /the special case
 - 12e /the refinement 12f /the alternate
- symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 1

The symptoms relevant to a *disk_crash* are:

software_process_lost
power_error
connector_problem
keyboard_locked
screen_blank
mouse_failing
hard_disk_access_denied

Please input the symptom names separated by blanks and ending with a period.

WHY do you need these symptoms for my *disk_crash*:

l: software_process_lost
l: power_error
l: connector_problem
l: keyboard_locked
l: screen_blank
l: mouse_failing
l: hard_disk_access_denied.

1 For my *disk_crash*, WHY do you need these symptoms:

software_process_lost
power_error
connector_problem
keyboard_locked
screen_blank
mouse_failing
hard_disk_access_denied

Your symptoms are needed since:

- > the symptom set [*software_process_lost,hard_disk_access_denied*] *enables disk_crash* so that it may be focused for an explanation.
- > the symptom set [*power_error*] *invalidates disk_crash* so that other observed symptoms may be examined for special case conditions.
- > the symptom set [*connector_problem*] provides *special conditions* for *disk_crash* so that it may override other observed symptoms that may invalidate it.
- > the symptom set [*hard_disk_access_denied*] is a *focusing* symptom set for the *disk_crash* so that it may be focused to a *disk_sector_problem*.
- > the symptom set [*keyboard_locked,screen_blank,mouse_failing*] is an alternate problem symptom set for the *disk_crash* so that it may be focused to a *processor_card_problem*.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 2

This was the relevant symptom not selected for the *disk\_crash*:  
*power\_error*

WHY do you NOT need this symptom for my *disk\_crash*:  
l: *power\_error*.

2 For my *disk\_crash*, WHY do you NOT need this symptom:  
*power\_error*

Your symptom was NOT needed since:  
--> the symptom set [*power\_error*] would have *invalidated disk\_crash* which  
would have required other observed symptoms to be examined for special case  
conditions.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: quit

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for *disk\_crash*
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: l: 5

Thank you for using RATIONALE running the acme workstation problem advisor.  
~~~~~

~~~~~  
Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice  
experienced  
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: !: novice

Your user level has been set to novice.

What type of problem have you been having with *acme* ?

software\_problem  
hardware\_problem  
communication\_problem  
power\_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: !: unknown

RATIONALE is considering **PROBLEM-SYMPTOMS** pairs to help determine your type of problem. For RATIONALE to focus on the type of problem you have, do one of the following:

If the symptoms listed below have all been observed, please type in the problem name, carriage return, and all.

If not all the listed symptoms have been observed, please type in the problem name, carriage return, and only those symptoms observed separated by blanks and ending with a period.

Possible SYMPTOMS of *software\_problem*

failed\_boot  
memory\_error  
software\_process\_lost

Possible SYMPTOMS of *hardware\_problem*

keyboard\_locked  
screen\_blank  
mouse\_failing  
memory\_error  
software\_process\_lost

Possible SYMPTOMS of *communication\_problem*

failed\_server\_access

Possible SYMPTOMS of *power\_problem*

fuse\_error  
on\_off\_switch\_ineffective

name of problem: l: software\_problem

Please input the symptom names separated by blanks and ending with a period.

symptoms of software\_problem: l: all.

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *software\_problem*

hardware\_error  
processor\_error  
software\_process\_lost  
hard\_disk\_access\_denied  
failed\_boot  
memory\_error

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of *software\_problem*:

l: processor\_error  
l: software\_process\_lost  
l: hard\_disk\_access\_denied  
l: failed\_boot.

Your *software\_problem* can be FOCUSED to give the following explanations:

The *disk\_crash* indicates that:

There is a problem with a *sector of the hard disk* of the workstation and it may require a hard disk overhaul. Run hard disk sector diagnostics and contact ACME service department.

The *processor\_software\_problem* indicates that:

There may be a problem with workstation *micro-processor software* and it may require reloading. Check processor diagnostics.

Your *software\_problem* has an *alternate* - a *hardware\_problem*.

The *hardware\_problem* indicates that:

There may be a problem with workstation *memory card* and it may require replacing. Run card diagnostics and replace it if it fails.

~~~~~

What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for *software_problem*
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: 1: 4

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *software_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *software_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *software_problem*, WHY did you conclude the *software_problem* or the subproblem I'm specifying below?
- 4 For my *software_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *software_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *software_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *software_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
 - 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are
 - 12a the relevant 12b /the enabling
 - 12c /the invalidating 12d /the special case
 - 12e /the refinement 12f /the alternate
- symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 3

These were the *related problems* and *subproblems* that were concluded for the *software_problem*

disk_crash
processor_software_problem
hardware_problem

Why did you conclude a: l: disk_crash

RATIONALE was able to conclude a *disk_crash* since:

--> your symptom set [*hard_disk_access_denied*]
allowed your *software_problem* to be focused to a *disk_crash* to match the
required degree of a minimum of 1.0

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 4

This was the related subproblem that was NOT concluded for the *software_problem*:
memory_problem

Why did you NOT conclude a: l: memory_problem

RATIONALE was NOT able to conclude a *memory_problem* since:

--> your observed symptom set, [*processor_error,software_process_lost,hard_disk_access_denied,failed_boot*] did NOT allow your
software_problem to be focused to a *memory_pproblem*. To match the
required degree of a minimum of 1.0 you also needed to observe the set,
[*memory_error*]

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: quit


~~~~~  
What would you like to do next? ·

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for software\_problem
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: 1: 5

Thank you for using RATIONALE running the acme workstation problem advisor.  
~~~~~

=====

Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice
experienced
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: 1: novice

Your user level has been set to novice.

What type of problem have you been having with *acme* ?

software_problem
hardware_problem
communication_problem
power_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: 1: unknown

RATIONALE is considering **PROBLEM-SYMPTOMS** pairs to help determine your type of problem. For RATIONALE to focus on the type of problem you have, do one of the following:

If the symptoms listed below have all been observed, please type in the problem name, carriage return, and all.

If not all the listed symptoms have been observed, please type in the problem name, carriage return, and only those symptoms observed separated by blanks and ending with a period.

Possible SYMPTOMS of *software_problem*

failed_boot

memory_error

software_process_lost

Possible SYMPTOMS of *hardware_problem*

keyboard_locked

screen_blank

mouse_failing

memory_error

software_process_lost

Possible SYMPTOMS of *communication_problem*

failed_server_access

Possible SYMPTOMS of *power_problem*

fuse_error

on_off_switch_ineffective

name of problem: !: **software_problem**

Please input the symptom names separated by blanks and ending with a period.

symptoms of software_problem: !: all.

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *software_problem*

hardware_error
processor_error
software_process_lost
hard_disk_access_denied
failed_boot
memory_error

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of *software_problem*:

!processor_error
!software_process_lost
!hard_disk_access_denied
!failed_boot.

•Your *software_problem* can be FOCUSED to give the following explanations:

The *disk_crash* indicates that:

There is a problem with a *sector of the hard disk* of the workstation and it may require a hard disk overhaul. Run hard disk sector diagnostics and contact ACME service department.

The *processor_software_problem* indicates that:

There may be a problem with workstation micro-processor software and it may require reloading. Check processor diagnostics.

Your *software_problem* has an *alternate* - a *hardware_problem*.

The *hardware_problem* indicates that:

There may be a problem with workstation *memory card* and it may require replacing. Run card diagnostics and replace it if it fails.

~~~~~

What would you like to do next?

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for *software_problem*
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: 4

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *software_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *software_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *software_problem*, WHY did you conclude the *software_problem* or the subproblem I'm specifying below?
- 4 For my *software_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *software_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *software_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *software_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
  - 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are
  - 12a the relevant   12b /the enabling
  - 12c /the invalidating   12d /the special case
  - 12e /the refinement   12f /the alternate
- symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 3

These were the *related problems* and *subproblems* that were concluded for the *software_problem*

disk_crash
processor_software_problem
hardware_problem

Why did you conclude a: l: **processor_software_problem**

RATIONALE was able to conclude a **processor_software_problem** since:

--> your symptom set [**failed_boot**] allowed your **software_problem** to be focused to a **processor_software_problem** to match the required degree of a minimum of 0.5

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 3

These were the related **problems** and **subproblems** that were concluded for the **software_problem**

disk_crash
processor_software_problem
hardware_problem

Why did you conclude a: l: **hardware_problem**

RATIONALE was able to conclude a **hardware_problem** since:

--> your symptom set [**software_process_lost**] allowed your **software_problem** to be focused to an **alternate** problem, a **hardware_problem**, to match the required degree of 0.5. This alternate problem may also be the source of your symptoms.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: quit

~~~~~

What would you like to do next?

1. re-select user level and re-display results
2. re-select problem
3. re-select symptoms for **software_problem**
4. ask questions about session results
5. quit RATIONALE running acme

user wants to: l: 5

Thank you for using RATIONALE running the acme workstation problem advisor.

~~~~~

~~~~~  
Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice
experienced
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: I: **novice**

Your user level has been set to novice.

What type of problem have you been having with *acme* ?

software_problem
hardware_problem
communication_problem
power_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: I: **hardware_problem**

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *hardware_problem*

power_error
keyboard_locked
screen_blank
mouse_failing
software_process_lost
memory_error

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of *hardware_problem*:

!:**keyboard_locked**
!:**screen_blank**
!:**mouse_failing**
!:**software_process_lost.**

Your *hardware_problem* can be FOCUSED to give the following explanations:

The *processor_card_problem* indicates that:

There may be a problem with workstation micro-processor card and it may require replacing. Run card diagnostics and replace if it fails.

The *memory_card_problem* indicates that:

There may be a problem with workstation memory card and it may require replacing. Run card diagnostics and replace it if it fails.

Your *hardware_problem* has an *alternate* - a *software_problem*.

The *software_problem* indicates that:

There may be a problem with ACME workstation software, run the diagnostics.

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for *hardware_problem*
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: !: 4
- 2

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *hardware_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *hardware_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *hardware_problem*, WHY did you conclude the *hardware_problem* or the subproblem I'm specifying below?
- 4 For my *hardware_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *hardware_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *hardware_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *hardware_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
- 11b What subproblems can the problem I'm specifying be focused to?

What symptoms and respective degrees of match are

12a the relevant	12b /the enabling
12c /the invalidating	12d /the special case
12e /the refinement	12f /the alternate

symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 3

These were the related *problems* and *subproblems* that were concluded for the *hardware_problem*

processor_card_problem
memory_card_problem
software_problem

Why did you conclude a: 1: processor_card_problem

RATIONALE was able to conclude a *processor_card_problem* since:

--> your symptom set [*keyboard_locked,screen_blank,mouse_failing*]
allowed your *hardware_problem* to be focused to a
processor_card_problem to match the required degree of a minimum of 1.0.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 5

These were the related *problems* and *subproblems* that were concluded for the *hardware_problem*

processor_card_problem
memory_card_problem
software_problem

For my *hardware_problem* HOW did you conclude a: 1: software_problem

RATIONALE was able to conclude a *software_problem* since:

--> your symptom set [*software_process_lost*] allowed your *hardware_problem* to
be focused to an *alternate* problem, a *software_problem* to match the required
degree of 0.5. This alternate problem may also be the source of your symptoms.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: quit

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for hardware_problem
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: 1: 5

Thank you for using RATIONALE running the acme workstation problem advisor.

~~~~~

=====

Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice
experienced
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: 1: **expert**

Your user level has been set to expert.

What type of problem have you been having with *acme* ?

software_problem
hardware_problem
communication_problem
power_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: 1: **hardware_problem**

RATIONALE is considering **RELEVANT SYMPTOMS** that help **FOCUS** your problem further. For **RATIONALE** to achieve this, do one of the following:

If the **RELEVANT** symptoms listed below have all been observed, please type all.

If not all the **RELEVANT** listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *hardware_problem*

power_error
keyboard_locked
screen_blank
mouse_failing
software_process_lost
memory_error

Please input the symptom names separated by blanks and ending with a period.

* **OBSERVED** symptoms of *hardware_problem* :

!keyboard_locked
!screen_blank
!mouse_failing
!software_process_lost
!memory_error.

Your *hardware_problem* can be **FOCUSED** to give the following explanations:

The *processor_card_problem* indicates:
Processor card failure. Trace and swap.

The *memory_card_problem* indicates:
Memory bank failure. Trace and call service to swap.

Your *hardware_problem* has an *alternate* - a *software_problem*.

The *software_problem* indicates:
Processor software bug or corruption needs tracing.
Memory software bug or corruption requires reload.

~~~~~

What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for hardware_problem
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: 1: 4

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *hardware_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *hardware_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *hardware_problem*, WHY did you conclude the *hardware_problem* or the subproblem I'm specifying below?
- 4 For my *hardware_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *hardware_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *hardware_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *hardware_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
 - 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are
 - 12a the relevant 12b /the enabling
 - 12c /the invalidating 12d /the special case
 - 12e /the refinement 12f /the alternate
- symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 3

These were the related *problems* and *subproblems* that were concluded for the
 hardware_problem
 processor_card_problem
 memory_card_problem
 software_problem

Why did you conclude a: l: memory_card_problem

RATIONALE was able to conclude a *memory_card_problem* since:
 --> your symptom set [*software_process_lost, memory_error*] allowed your
hardware_problem to be focused to a *memory_card_problem* to match the
 required degree of a minimum of 0.5.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 5

These were the related *problems* and *subproblems* that were concluded for the
hardware_problem

processor_card_problem

 memory_card_problem
 memory_bank_problem

 software_problem
 processor_software_problem
 memory_problem

For my *hardware_problem* HOW did you conclude a:
 l: memory_bank_problem

RATIONALE was able to conclude a *memory_bank_problem* since:
 --> your *memory_card_problem* was a focusing problem for your
hardware_problem
 --> your symptom set [*memory_error*] allowed your *memory_bank_problem* to be
 enabled to match the required degree of 0.5

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 5

These were the related *problems* and *subproblems* that were concluded for the *hardware_problem*

processor_card_problem

memory_card_problem

memory_bank_problem

software_problem

processor_software_problem

memory_problem

For my *hardware_problem* HOW did you conclude a:

1: *memory_problem*

RATIONALE was able to conclude a *memory_problem* since:

--> your symptom set [*memory_error*] allowed your *memory_problem* to be *enabled* to the required degree of 0.5

--> your *software_problem* was an *alternate* problem for your *hardware_problem*

→ your symptom set [*memory_error*] allowed your *software_problem* to be focused to a *memory_problem* to match the required degree of a minimum of 1.0

~~~~~

B3: Hypothetical Question Explanations

Continuing with the session immediately above in the event explanations of section B2.

~~~~~

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *hardware_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *hardware_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *hardware_problem*, WHY did you conclude the *hardware_problem* or the subproblem I'm specifying below?
- 4 For my *hardware_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *hardware_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *hardware_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *hardware_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
  - 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are
  - 12a the relevant   12b /the enabling
  - 12c /the invalidating   12d /the special case
  - 12e /the refinement   12f /the alternate
- symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 6

To remind you, your *previously observed symptoms* are:

keyboard_locked
screen_blank
mouse_failing
software_process_lost
memory_error

The *symptoms relevant to a hardware_problem* are:

power_error
keyboard_locked
screen_blank
mouse_failing
software_process_lost
memory_error

Please input the symptom names separated by blanks and ending with a period.

WHAT-IF these symptoms were added to my above ones for *hardware_problem* :

l: **power_error.**

6 What-if this symptom was also true for the *hardware_problem* ?
power_error

If **power_error** was added to the previous symptoms for the *hardware_problem*,
Your *hardware_problem* can be focused to give the following explanations:

The *hardware_problem* indicates that:
Workstation hardware failure needs tracing.

Your *hardware_problem* has an *alternate* - a *software_problem* .

The *software_problem* indicates that:
Processor software bug or corruption needs tracing.
Memory software bug or corruption requires reload.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: quit


~~~~~  
What would you like to do next?

1. re-select user level and re-display results
  2. re-select problem
  3. re-select symptoms for hardware\_problem
  4. ask questions about session results
  5. quit RATIONALE running acme
- user wants to: 1: 5

Thank you for using RATIONALE running the acme workstation problem advisor.  
~~~~~

~~~~~  
Welcome to *acme* running in RATIONALE!

What type of user are you ?

novice  
experienced  
expert

Please type in novice if you are a new user, experienced if you are an occasional user and expert if you are very familiar with the system.

user level: 1: novice

Your user level has been set to novice.

What type of problem have you been having with *acme* ?

software\_problem  
hardware\_problem  
communication\_problem  
power\_problem

Please type in the name of the problem if you know it. If you do not know it, type in unknown.

type of problem: 1: unknown

RATIONALE is considering **PROBLEM-SYMPTOMS** pairs to help determine your type of problem. For RATIONALE to focus on the type of problem you have, do one of the following:

If the symptoms listed below have all been observed, please type in the problem name, carriage return, and all.

If not all the listed symptoms have been observed, please type in the problem name, carriage return, and only those symptoms observed separated by blanks and ending with a period.

Possible SYMPTOMS of *software\_problem*

failed\_boot

memory\_error

software\_process\_lost

Possible SYMPTOMS of *hardware\_problem*

keyboard\_locked

screen\_blank

mouse\_failing

memory\_error

software\_process\_lost

Possible SYMPTOMS of *communication\_problem*

failed\_server\_access

Possible SYMPTOMS of *power\_problem*

fuse\_error

on\_off\_switch\_ineffective

name of problem: l: **software\_problem**

Please input the symptom names separated by blanks and ending with a period.

symptoms of software\_problem: l: **all.**

RATIONALE is considering **RELEVANT SYMPTOMS** that help FOCUS your problem further. For RATIONALE to achieve this, do one of the following:

If the RELEVANT symptoms listed below have all been observed, please type all.

If not all the RELEVANT listed symptoms have been observed, please type in only those symptoms observed separated by blanks and ending with a period.

RELEVANT SYMPTOMS of *software\_problem*.

hardware\_error  
processor\_error  
software\_process\_lost  
hard\_disk\_access\_denied  
failed\_boot  
memory\_error

Please input the symptom names separated by blanks and ending with a period.

OBSERVED symptoms of *software\_problem*:

!processor\_error  
!software\_process\_lost  
!hard\_disk\_access\_denied  
!failed\_boot.

Your *software\_problem* can be FOCUSED to give the following explanations:

The *disk\_crash* indicates that:

There is a problem with a *sector of the hard disk* of the workstation and it may require a hard disk overhaul. Run hard disk sector diagnostics and contact ACME service department.

The *processor\_software\_problem* indicates that:

There may be a problem with workstation *micro-processor software* and it may require reloading. Check processor diagnostics.

Your *software\_problem* has an *alternate* - a *hardware\_problem*.

The *hardware\_problem* indicates that:

There may be a problem with workstation *memory card* and it may require replacing. Run card diagnostics and replace it if it fails.

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for *software_problem*
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: !: 4

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *software_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *software_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *software_problem*, WHY did you conclude the *software_problem* or the subproblem I'm specifying below?
- 4 For my *software_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *software_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *software_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *software_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
 - 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are
 - 12a the relevant 12b /the enabling
 - 12c /the invalidating 12d /the special case
 - 12e /the refinement 12f /the alternate
- symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 6

To remind you, your *previously observed symptoms* are:

processor_error
software_process_lost
hard_disk_access_denied
failed_boot

The *symptoms relevant to a software_problem* are:

hardware_error
processor_error
software_process_lost
hard_disk_access_denied
failed_boot
memory_error

Please input the symptom names separated by blanks and ending with a period.

WHAT-IF these symptoms were added to my above ones for *software_problem* :

!hardware_error
!memory_error.

6 What-if these symptoms were also true for the *software_problem*?

hardware_error
memory_error

If the *symptoms hardware_error and memory_error* were added to the previous ones for *software_problem*,

Your *software_problem* can be FOCUSED to give the following explanations:

The *disk_crash* indicates that:

There is a problem with a *sector of the hard disk* of the workstation and it may require a hard disk overhaul. Run hard disk sector diagnostics and contact ACME service department.

The *processor_software_problem* indicates that:

There may be a problem with workstation micro-processor software and it may require reloading. Check processor diagnostics.

The *memory_problem* indicates that:

There may be a problem with the *memory partition*. Try re-loading a new partition to work in.

Your *software_problem* has an *alternate* - a *hardware_problem*.

The *hardware_problem* indicates that:

There may be a problem with workstation *memory card* and it may require replacing. Run card diagnostics and replace it if it fails.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 7

subproblem: l: **hardware_problem**

The *symptoms relevant* to a *hardware_problem* are:

power_error
 keyboard_locked
 screen_blank
 mouse_failing
 software_process_lost
 memory_error

Please input the symptom names separated by blanks and ending with a period.

WHAT-IF these symptoms are observed for *hardware_problem* :

l:keyboard_locked
 l:screen_blank
 l:mouse_failing
 l:software_process_lost
 l:memory_error.

7 What-if the *hardware_problem* replaced the *software_problem* with these new symptoms?

keyboard_locked
screen_blank
mouse_failing
software_process_lost
memory_error

Your *hardware_problem* can be FOCUSED to give the following explanations:

The *processor_card_problem* indicates that:

There may be a problem with workstation micro-processor card and it may require replacing. Run card diagnostics and replace if it fails.

The *memory_card_problem* indicates that:

There may be a problem with a *bank of chips* on the workstation memory card and they may require replacing. Run memory bank diagnostics and call for service if it fails tests.

Your *hardware_problem* has an *alternate* - a *software_problem*.

The *software_problem* indicates that:

There may be a problem with workstation *micro-processor software* and it may require reloading. Check processor diagnostics.

There may be a problem with workstation *memory software* and it may require reloading. Run memory diagnostics.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 7

subproblem: 1: *communication_problem*

The *symptoms relevant* to a *communication_problem* are:

software_error
hardware_error
connector_error
memory_access_denied
failed_server_access

Please input the symptom names separated by blanks and ending with a period.

WHAT-IF these symptoms are observed for *communication_problem*:

1:*software_error*
1:*hardware_error*
1:*connector_error*
1:*memory_access_denied*
1:*failed_server_access*.

7 What-if the *communication_problem* replaced the *software_problem* with these new symptoms?

software_error
hardware_error
connector_error
memory_access_denied
failed_server_access

Your *communication_problem* can be FOCUSED to give the following explanations:

The *server_problem* indicates that:

The problem with workstation-server communication may be due to a *server-ethernet communication problem*. Contact administrator.

Your *communication_problem* has an *alternate* - a *memory_problem*.

The *memory_problem* indicates that:

There may be a problem with a *segment of workstation memory*. Run memory diagnostics and contact ACME service.

B4: Ability Question Explanations

Continuing with the session immediately above in the hypothetical explanations of section B3.

~~~~~

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *software_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *software_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *software_problem*, WHY did you conclude the *software_problem* or the subproblem I'm specifying below?
- 4 For my *software_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *software_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *software_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *software_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
- 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are

12a the relevant	12b /the enabling
12c /the invalidating	12d /the special case
12e /the refinement	12f /the alternate

 symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 8

subproblem: 1: **disk_crash**

8 How do you *deduce* a *disk_crash* ?

The **disk_crash** is deduced by matching your observed symptoms against the required enabling symptoms, [software_process_lost,hard_disk_access_denied] to get the specified degree of match of 0.5.

Once a **disk_crash** is enabled to the specified degree, if its invalidating symptom, power_error is observed, it is found to satisfy the invalidating degree of match of 1.0.

If the **disk_crash** is found invalid, the observed symptoms are checked for its special case symptom, connector_problem specified for a degree of match of 1.0. If the symptom is observed, it overrides the invalidation condition and **disk_crash** is deduced.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 8

subproblem: 1: **software_problem**

8 How do you *deduce* a *software_problem* ?

The **software_problem** is deduced by matching your observed symptoms against the required enabling symptoms, [failed_boot,memory_error,software_process_lost] to get the specified degree of match of 0.3.

Once a **software_problem** is enabled to the specified degree, if its invalidating symptom, hardware_error is observed, it is found to satisfy the invalidating degree of match of 1.0.

If the **software_problem** is found invalid, the observed symptoms are checked against its special case symptoms, [failed_boot,processor_error,memory_error] for a specified degree of match of 0.3.

Once a **software_problem** is enabled and found valid (by checking the special case symptoms if necessary), it is deduced.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 9

subproblem: 1: **software_problem**

9 How do you deduce the *alternate problem* for a *software_problem* ?

To deduce the alternate subproblem, hardware_problem, for a software_problem, the alternate's required symptoms, [memory_error, software_process_lost] are checked against your observed symptoms. If a match of at least 0.5 is achieved, then the alternate problem is deduced and refined as far as your observed symptoms allow.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 9

subproblem: 1: **hardware_problem**

9 How do you deduce the *alternate problem* for a *hardware_problem* ?

To deduce the alternate problem, software_problem, for a hardware_problem the alternate's required symptoms, [memory_error, software_process_lost] are checked against your observed symptoms. If a match of at least 0.5 is achieved, then the alternate problem is deduced and refined as far as your observed symptoms allow.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 10

subproblem: 1: **software_problem**

10 How do you *refine* a *software_problem* ?

A software_problem can be refined as follows:

- > to refine to subgoal disk_crash the refinement symptom,
hard_disk_access_denied is to be observed to satisfy a degree of match of 1.0.
- > to refine to subgoal processor_software_problem the refinement symptoms,
[failed_boot, memory_error] are to be observed to satisfy a degree of match of 0.5.
- > to refine to subgoal memory_problem the refinement symptom, memory_error is to be observed to satisfy a degree of match of 1.0.

B5: Factual Question Explanations

Continuing with the session immediately above in the ability explanations of section B4.

~~~~~

The following are the *session explanation questions* that RATIONALE can answer. Simply input the number associated with the question and input the specified information you are prompted for.

EVENT QUESTIONS:

- 1 For my *software_problem*, WHY do you need the symptoms I'm specifying below?
- 2 For my *software_problem*, WHY do you NOT need the symptoms I'm specifying below?
- 3 For my *software_problem*, WHY did you conclude the *software_problem* or the subproblem I'm specifying below?
- 4 For my *software_problem*, WHY did you NOT conclude the subproblem I'm specifying below?
- 5 For my , HOW did you deduce the *software_problem* or the subproblem I'm specifying below?

HYPOTHETICAL QUESTIONS:

- 6 What-if the symptoms I'm specifying were added to my previous symptoms for the *software_problem* ?
- 7 What-if the subproblem I'm specifying replaced the *software_problem* with the symptoms I'm also specifying below?

ABILITY QUESTIONS:

- 8 How do you deduce the subproblem I'm specifying below?
- 9 How do you deduce the alternate subproblem for the subproblem I'm specifying below?
- 10 How do you refine the subproblem I'm specifying below?

FACTUAL QUESTIONS:

- 11a What subproblems can the system handle?
- 11b What subproblems can the problem I'm specifying be focused to?
What symptoms and respective degrees of match are

12a the relevant	12b /the enabling
12c /the invalidating	12d /the special case
12e /the refinement	12f /the alternate

 symptoms and degrees of match for the subproblem I'm specifying below.

Enter your explanation number choice or enter QUIT to return to the main menu.

~~~~~

explanation menu number: l: 10

subproblem: l: **power_problem**

10 How do you *refine* a *power_problem* ?

A *power_problem* does not have any refinement subgoals and as a result is not refined.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 11a

11a What *subproblems* can the system handle?

RATIOANLE running the *acme advisor*, can handle the following top level problems:

software_problem

hardware_problem

communication_problem

power_problem

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 11b

subproblem: l: **software_problem**

11b What subproblems can a *software_problem* be *focused* to?

A *software_problem*, depending on symptoms, can be focused to any one or all of the following subproblems:

disk_crash

processor_software_problem

memory_problem

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 11b

subproblem: 1: disk_crash

11b What subproblems can a *disk_crash* be *focused* to?

A *disk_crash* , depending on symptoms, can be focused to a *disk_sector_problem* .

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: 1: 11b

subproblem: 1: disk_sector_problem

11b What subproblems can a *disk_sector_problem* be *focused* to?

A *disk_sector_problem* , has *no subproblems* to be focused further to.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 11b

subproblem: l: memory_problem

11b What subproblems can a *memory_problem* be focused to?

A *memory_problem*, depending on symptoms, can be focused to any one or all of the following subproblems:

memory_segment_problem

memory_partition_problem

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 11b

subproblem: l: memory_partition_problem

11b What *subproblems* can a *memory_partition_problem* be focused to?

A *memory_partition_problem*, has *no* subproblems to be focused further to.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12a

subproblem: l: software_problem

12a What *symptoms* are the *relevant* ones for a *software_problem* ?

The symptoms relevant to a software_problem are:

- hardware_error
- processor_error
- software_process_lost
- hard_disk_access_denied
- failed_boot
- memory_error

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12a

subproblem: l: disk_crash

12a What *symptoms* are the *relevant* ones for a *disk_crash* ?

The symptoms relevant to a disk_crash are:

- software_process_lost
- power_error
- connector_problem
- keyboard_locked
- screen_blank
- mouse_failing
- hard_disk_access_denied


~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12b

subproblem: l: disk_crash

12b What *symptoms and respective degree of match enable a disk_crash* ?

The symptoms that enable a disk_crash are:

software_process_lost

hard_disk_access_denied

and they must match to the degree of 0.5.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12c

subproblem: l: disk_crash

12c What *symptoms and respective degree of match invalidate a disk_crash* ?

The symptom that invalidates a disk_crash is a power_error
and it must match to the degree of 1.0.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12d

subproblem: l: disk_crash

12d What *symptoms and respective degree of match are the special case symptoms* that override invalidation for a *disk_crash* ?

The symptom that overrides the invalidation of the disk_crash is a connector_problem and it must match to the degree of 1.0.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12e

subproblem: l: disk_crash

12e   What *symptoms and respective degree are required to refine*
      a *disk_crash* ?

The symptom that triggers the refinement subproblem, disk_sector_problem for the
disk_crash is a hard_disk_access_denied
and it must match to the degree of 1.0.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12f

subproblem: l: disk_crash

12f What *symptoms and respective degree of match trigger the alternate*
 subproblem of a *disk_crash* ?

The symptoms that trigger the alternate subproblem, processor_card_problem of the
disk_crash are:

 keyboard_locked

 screen_blank

 mouse_failing

and they must match to the degree of 1.0.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12a

subproblem: l: memory_partition_problem

12a What *symptoms are the relevant* ones for a *memory_partition_problem* ?

The symptoms relevant to a memory_partition_problem are:

failed_boot
power_error
memory_error
memory_access_denied

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12b

subproblem: l: memory_bank_problem

12b What *symptoms and respective degree of match enable* a
memory_bank_problem ?

The symptoms that enable a memory_bank_problem are:

memory_error
memory_access_denied

and they must match to the degree of 0.5.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12c

subproblem: l: **memory_segment_problem**

12c   *What symptoms and respective degree of match invalidate a
memory_segment_problem ?*

The symptom that invalidates a memory_segment_problem is a power_error and it must match to the degree of 1.0.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12d

subproblem: l: **memory_segment_problem**

12d *What symptoms and respective degree of match are the special case
symptoms that override invalidation for a memory_segment_problem ?*

The memory_segment_problem has no special case symptoms to override invalidation.

~~~~~

Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12e

subproblem: l: **memory_problem**

12e   *What symptoms and respective degree are required to refine a
memory_problem ?*

Several refinements are possible for a memory_problem:

--> to refine to subgoal memory_segment_problem the refinement symptom,
memory_access_denied is to be observed with a degree of match of 1.0.

--> to refine to subgoal memory_partition_problem the refinement symptom, failed_boot is
to be observed with a degree of match of 1.0.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: 12f

subproblem: l: memory_bank_problem

12f What *symptoms and respective degree of match trigger the alternate*
subproblem of a *memory_bank_problem* ?

Several alternate refinements are possible for a memory_bank_problem:

--> to refine to subgoal memory_segment_problem the refinement symptoms,
[memory_error,memory_access_denied] are to be observed with a degree of match
of 1.0.

--> to refine to subgoal memory_partition_problem the refinement symptom, failed_boot is
to be observed with a degree of match of 1.0.

~~~~~  
Enter your explanation number choice or enter QUIT to return to the main menu.

explanation menu number: l: quit

~~~~~  
What would you like to do next?

1. re-select user level and re-display results
 2. re-select problem
 3. re-select symptoms for software_problem
 4. ask questions about session results
 5. quit RATIONALE running acme
- user wants to: l: 5

Thank you for using RATIONALE running the acme workstation problem advisor.

~~~~~  
~~~~~

END

06-07-89

FIN