AI TR•581

# A MODEL FOR DELIBERATION, ACTION, AND INTROSPECTION

JON DOYLE

MAY 1980

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

# A Model for Deliberation, Action, and Introspection

by

Jon Doyle

Massachusetts Institute of Technology
May 1980

This report reproduces a dissertation submitted on May 12, 1980 to the Department of Electrical Engineering and Computer Science of the Massachusetts Institute of Technology in partial fulfillment of the requirements of the degree of doctor of philosophy.

# A Model for Deliberation, Action, and Introspection

by

Jon Doyle

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 1980 in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy in
Artificial Intelligence

## ABSTRACT

This thesis investigates the problem of controlling or directing the reasoning and actions of a computer program. The basic approach explored is to view reasoning as a species of action, so that a program might apply its reasoning powers to the task of deciding what inferences to make as well as deciding what other actions to take. A design for the architecture of reasoning programs is proposed. This architecture involves self-consciousness, intentional actions, deliberate adaptations, and a form of decision-making based on dialectical argumentation. A program based on this architecture inspects itself, describes aspects of itself to itself, and uses this self-reference and these self-descriptions in making decisions and taking actions. The program's mental life includes awareness of its own concepts, beliefs, desires, intentions, inferences, actions, and skills. All of these are represented by self-descriptions in a single sort of language, so that the program has access to all of these aspects of itself, and can reason about them in the same terms.

Thesis Supervisor: Gerald Jay Sussman

Title: Associate Professor of Electrical Engineering

# CONTENTS

# FIGURES

[illegible]
[illegible]
[illegible]

## Acknowledgments

I thank Gerald Sussman, my thesis advisor, and my readers Peter Szolovits, Drew McDermott, and Marvin Minsky for their constant encouragement, criticism, and advice.

I thank Johan de Kleer for serving as a *de facto* reader for this thesis, and for continually valuable discussions on many topics. Many of the explanations in this thesis resulted from his comments.

I thank Joseph Schatz for more advice, stimulation, and pleasure than I can express, delivered with gentle wit and style. I would not be where I am without his concern.

I thank my family, Leo Doyle, Marilyn Doyle, Paul Doyle, Lynn Doyle, and Peter Doyle for teaching me how to live, to enjoy and excite, and to learn, and for moral and financial support whose importance to me cannot be underestimated.

I thank my aunt Cloe Doyle, for early teaching me about negative numbers and other parts of mathematics. It still comes in handy.

I thank Marcella Boffa, Kelly Martino, and John Baker for offering me the life I have forsaken.

I thank my friends who built my social interests: John and Sharon Cullen, Rebecca Schatz, Michael Loui, Shelly Lieber, Donald Petersen, Ronald Pankiewicz, and Marilyn Matz.

I thank all those people with whom I have discussed matters of interest. I'm afraid that, unlike my programs, I do not always recall where ideas came from, and I apologize to and cherish the contributions of all those whose suggestions I have considered. I hope they take my adoption of (or disagreement with) their ideas understandingly. In addition to the above, I especially thank Howard Shrobe, David McAllester and Richard Weyhrauch.

I thank my colleagues and the staff at the MIT Artificial Intelligence Laboratory and the Laboratory for Computer Science for their support in ideas, services, machines, and libraries.

I thank Daniel Carnese and Randall Davis for valuable comments on this thesis, and Gerald Roylance for teaching me how to draw the diagrams.

I thank Doubleday and Company for their permission to reprint the excerpt from *The Sot-Weed Factor* by John Barth, which is Copyright 1960, 1967 by John Barth.

Finally, I thank the Fannie and John Hertz Foundation for supporting me during my entire graduate career with a graduate fellowship. It was invaluable.

### He wishes for the cloths of heaven

by

W. B. Yeats

Had I the heavens' embroidered cloths,
Enwrought with golden and silver light,
The blue and the dim and the dark cloths
Of night and light and the half-light,
I would spread the cloths under your feet:
But I, being poor, have only my dreams;
I have spread my dreams under your feet;
Tread softly because you tread on my dreams.

# CHAPTER 1

# INTRODUCTION

Self-reverence, self-knowledge, self-control,
These three alone lead life to sovereign power.
Alfred, Lord Tennyson, *OEnone*

Know prudent cautious self-control is wisdom's root.
Robert Burns, *A Bard's Epitaph*

The woman that deliberates is lost.
Joseph Addison, *Cato*

But a self-controlled man is of a different sort:
he follows right reason.
Aristotle, *Nichomachian Ethics*

This thesis investigates the problem of controlling or directing the reasoning and actions of a computer program.[1] The basic approach explored is to view reasoning as a sort of action, and to have the program apply its reasoning powers to the task of deciding what inferences to make as well as deciding what other actions to take. This problem of controlling reasoning is important because information is often communicated between man and man, and eventually, it can be expected, between man and machine, as facts which offer little guidance as to what inferences should be drawn from them. Much experience and many theoretical studies have proven that the general problem of drawing particular conclusions from purely factual information is hopelessly intractable.[2] These lessons show that inference cannot always be treated as an automatic procedure, but sometimes must be accorded all the careful consideration given to

[1] Following current usage, this thesis uses the phrase "the program" to abbreviate some phrase resembling "the machine produced as a state-configuration of a LISP-implementing computer as described by the program text." Several writers, such as Fodor [1978], Putnam [1978], and Searle [1980] have pointed out that a program, a formal system, cannot be said to have a psychology, in contrast to apparent claims made by some AI researchers. Some of this disagreement might result from the unconscious use of an abbreviation of the above form on the part of some of the participants. We do not attempt to adjudicate this debate, nor to make rigorous the sense of the above abbreviation. Those tasks are left for others. (Brian Smith is engaged in such an enterprise.) In particular, this thesis avoids the problem of how the program is to be equipped with sensors and effectors so as to perceive and have power over its environment.

[2] See for example [Green 1969] and [Rabin 1974].

other actions. To overcome this difficulty, this thesis attempts to find ways of stating and using facts about how other facts should enter into reasoning. The proposed solution, that of a program which reasons about its own reasoning, is of considerable generality.

This chapter consists of several sections. The first section presents what the fundamental argument for the approach, and this argument introduces the fundamental ideas of and constraints on the proposed solution. The second and third sections of the chapter sketch the structure and operation of the program described in the following chapters, and give a guide to reading those chapters. The fourth section presents examples of the ideas in practice. The fifth section discusses the implementation. The sixth section attempts to motivate the approach with yet another argument, this time by showing how one might be lead to the proposed approach purely from considerations of what must be computed and what is necessary to allow its computation. The final section of the chapter sketches the relation of this thesis to other works.

Readers concerned with how to use the techniques developed in this thesis are cautioned in two ways. First, most of the techniques employed in traditional AI programs, such as problem reduction problem solving, planning, searching, backtracking, learning, context switching, etc., occur only in Chapter 6 as applications. The bulk of the thesis is devoted to foundational tools by which these traditional techniques may be used deliberately, and consequently, may be explained by the program itself. Second, the techniques apparently require an unusually large overhead in time, space, and notation. Sections 1.5 and 1.6 explain why this overhead must be accepted to build intelligent machines.

The reader will find that many of the techniques explained in the thesis bear a certain similarity to mechanisms which some commonsense truisms ascribe to the workings of the human mind. This similarity results solely from my use of these truisms together with informal personal introspection to inform my development of the proposed techniques from primarily computational considerations. I make no claim that the human mind employs similar mechanisms. I merely attempt to motivate and explain these techniques with common ideas about human behavior, since humans are currently the best-known concrete model of intelligent behavior. Specifically, I try to indicate how a number of mechanisms originally developed for rather technical tasks, including the design, synthesis, and analysis of electronic circuits and computer programs, can be combined and organized to capture common-sense reasoning as well as highly specific technical problem-solving. I frequently motivate my suggestions with common-sense examples, as they are significantly easier to convey, but I hope it is clear that these mechanisms also suffice for the traditional technical tasks. Briefly put, I view technical reasoning as a subcategory of general reasoning, and a more tractable one at that. But though I try to capture a number of familiar human reasoning patterns, no claim should be inferred that the mechanisms I propose are the only such mechanisms, or that they are those used by humans.

## 1.1 The Fundamental Argument

This section attempts to motivate the proposed approach through a series of propositions, loosely called an argument, which express general criteria for judging proposed organizations for intelligent machines. These propositions capture certain characteristics of intelligent existence, characteristics which significantly constrain proposed organizations of intelligent machines. Because of their generality, the propositions are presented with both motivation and morals. We first motivate the argument with a parable in primitive human terms, and follow the parable with the propositions of the fundamental argument, annotating each proposition with a moral about how intelligent machines should be organized. The paragraphs of the parable parallel the relevant propositions of the argument.

### 1.1.1 The Parable

I always adapt. I put on skins in winter, follow the game as it migrates, and run from rolling rocks, falling trees, and charging behemoths.

Sometimes I must change things, sometimes myself. As the years grow colder, I move towards the south. As the land turns barren, I train myself to be forever alert to take advantage of the fleeting, infrequent opportunities for food. Sometimes I must change both my surroundings and myself. When the plains and plateau became infested with dangerous beasts, I moved to live in the cliff, and trained myself to be a good and careful rock climber.

To avoid mistakes, I think carefully before acting. Normally I go unarmed, but if I thoughtlessly walk unarmed into the forest, without first reflecting on what I am doing, I am likely later to meet an unpleasant end with lions, tigers, or bears. When I decided to build a shelter on the ground without first reflecting on the decisions that had to be made and the order in which they should have been made, I wasted much of my own effort, and that of my sons as well that I asked to help, for I thought about the various clays I know of and of where they can be found and how they might be carried, before I turned to the question of where to build and realized that the best location was in the flood plain, where our crops would be nourished, where we would spend two days a year waiting out the flood uplands or in a tree, but where adobe would be a useless waste of effort compared with a thatched hut. And when I tell my son what to do, I must think of what he knows of my plans for the hut, lest I say things he does not understand, and of how to say the orders, for he is proud, easy to anger, and I am not as strong as I once was.

I have so many decisions to make. The farm has done well, but now I grow old and must divide it among my children. How should I do this? I can divide the lands in equal measures, some good and bad land to each, or I can divide it into the better and poorer fields, or I can split some of the larger fields into parts but not the smaller ones, or I can split them so that each has access to the stream, or I can divide them as the children request me to do.

There are so many complications. If I leave some of the good land and some of the bad land to each, then that will be fairest. But if some of them do poorly on their own, the lot of their brothers may not be sufficient to tide everyone over. But if I give the easiest lands to the weakest, the others will drive them out when I am gone, and they will have nothing. But even if I am fair, the eldest and the strongest will demand the largest and best lands. Perhaps I can give them just enough more to keep them from attacking the others. But I already promised the apple tree to the first daughter, and I must give extra land to the second son, who will care for my mate. But the strongest should be on the perimeter, to ward off invaders. What should I do? Should I do what is fair? What is safest from the whims of nature? What is most likely to be respected by the children? What I have promised each I would do? What will provide for my mate, as my mate provided for me? What is safest from enemies?

Woe, woe, sometimes I just can't help things. The invaders came, and now we are their slaves. Our women they took as mates. The mate of my first son, who had come from afar and spoke of gods and laws, would not submit to them, and they killed her. My daughters, whom she had convinced of these gods, instead of following her renounced the laws and went with the invaders, so they still live.

I hate this slavery. Why should the invaders rule? They are no better than us, and if we had invaded them first with similar surprise, we would be the masters and they the slaves. Why did this happen to us, and not to the other neighbors of the invaders? The dead one said that the gods put us here, but why should they do that? Why do the gods exist to torment us so? But if they do not exist, then the dead one is no more, not in this marvelous land she talked of. Does that await me too?

### 1.1.2 The Propositions

*1. The world continually changes, so to survive, we must always adapt.*

Intelligent machines should adapt to newly acquired information and to new demands placed on their operation by their users or co-workers.

*2. To adapt, we must act either to change our surroundings or ourselves.*

Intelligent machines should be able to modify their own organization and behavior as well as take physical actions.

*3. To act effectively, we must think about what to do, including thinking itself, so that we plan and reflect on our inferences as well as other actions.*

Intelligent machines should reason about their own organization and reasoning, as well as "external" domains, and plan complex "internal" activities (such as difficult decisions, comprehensive database searches, etc.) as well as complex external activities. Even parallel computations, however useful they might be in some ways, cannot relieve the need to make some consequential decisions serially.

*4. The most difficult problem in thinking about what to do is deciding between the many possible courses of action.*

Intelligent machines should when necessary explicitly consider decisions about which inference rule or procedure to apply next, where to look for some fact in the database, etc. so as to avoid combinatorially explosive searches.

*5. Decision-making, in turn, is dominated by the many incomparable sorts of reasons for or values of possible actions, which stem from a sectioning of the world into many subdomains, each with its own* *concerns and values. These incomparable reasons make decision-making a question of right (in one subdomain) vs. right (in another subdomain), not right vs. wrong.*

Intelligent machines should reason about their reasons for taking actions, to see if these reasons are of comparable types, or if they have exceptions in the current situation. Intelligent machines should not use decision-making techniques which force all reasons into a total order, as do most numerical weighting schemes.

*6. Further, our abilities are limited, which sometimes prevents our adapting by conserving or otherwise controlling our surroundings, so we must either always be able and ready to change any aspect of ourselves, or be willing to accept injury when we do not change.*

Intelligent machines should be able to deliberately change any of their database facts, procedures, etc., whether "built-in" or not.

*7. (The great joke is that though we need both self-consciousness and self-adaptiveness to survive, in combination these abilities shock us with realizations of both our own absurdity (why should we exist?) and the possibility of our own death (we might not exist!).)*

Intelligent machines should matter to themselves. They should have values initially built in so that they do not lightly change themselves into non-existence. They should choose their actions with responsibility for their own survival or other conditions that they are charged with maintaining.

## 1.2 Outline of the Approach

I think that many philosophers secretly harbor the view that there is something deeply (i.e., conceptually) wrong with psychology, but that a philosopher with a little training in the techniques of linguistic analysis and a free afternoon could straighten it out.
Jerry Fodor, *Psychological Explanation*

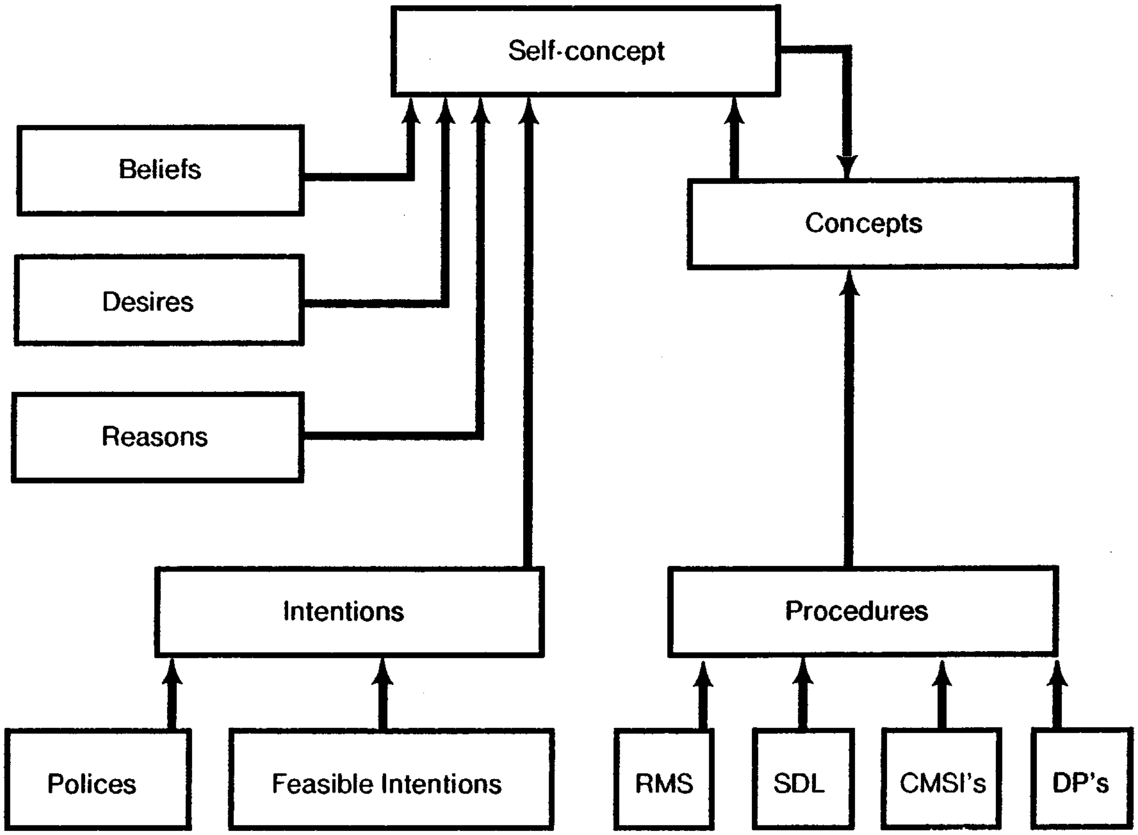
There's no art to find the mind's construction in the face.
William Shakespeare, *Macbeth*

Good Lord, what is man! for as simple he looks,
Do but try to develop his hooks and his crooks,
With his depths and his shallows, his good and his evil,
All in all, he's a problem must puzzle the devil.
Robert Burns, *Sketch: inscribed to C. J. Fox*

Motivated by the preceding ideas, this thesis sketches the basic computational structure of a conscious, adaptive reasoning program which we call SEAN. The program inspects itself, describes aspects of itself to itself, and uses these self-references and self-descriptions in making decisions and taking actions. The program's mental life includes awareness of its own concepts, beliefs, desires, intentions, inferences, values, past actions, and skills. These are realized by self-descriptions in a single sort of language, so that through self-reference the program has access to all of these aspects of itself, and can reason about them in their own language.[3]

The *concepts* of the program are each realized as (roughly) a named set of axioms in a formal logical language. The language is a variant of the first-order predicate calculus, but that detail is inessential. The key property of this representation is that the logical theories can themselves be referred to by other theories. This allows the program to employ statements about, for example, how its concept **horse** is related to its concept **animal**. In fact, the program itself is such a logical theory, and its language includes a name for itself. This allows the program to employ other statements that, for example, use the program's name for itself to refer to properties of the program as a whole, such as whether some possible belief is consistent with all of its current beliefs. This meta-theoretical approach allows some classical

[3] Figure 1 presents the overall program structure as described below.



RMS is the Reason Maintenance System
SDL is the Structured Description Language
CMSI's are current mental state interpreters
DP's are deliberation procedures

Figure 1

The Overall Program Structure

problems of representation to be attacked in effective ways, and allows reasoning about concepts in hierarchical levels of detail. Since the concepts are themselves objects to which the program can refer, the program can reason about whether or not to pursue the internal structure of a concept's subconcepts during information retrieval. This means that the program can ignore unnecessary details of its concepts when desired, and that the reasoner can be self-applied to the database retrieval task when necessary to avoid blind searches.

These concepts are then used in other logical theories to realize the mental attitudes of *beliefs*, *desires*, and *intentions*. These attitudes use a concept as their "propositional content." They are more than just the concepts embodying their propositional content, for they also include information used by the program in treating them as attitudes. These attitudes are also logical theories, but ones which are treated in special ways by the program, namely as beliefs, desires, and intentions.

The most important auxiliary information included by the program in attitudes over their content concerns the reasons for the attitudes. The program records its actions by adding statements describing them to itself. Inferences are sorts of actions, and hence are also recorded. Each attitude includes mention of these recorded inference steps, which we call the *reasons* for the attitude. Common usage normally uses the term "reason" to refer to an antecedent attitude acted on in an inference step, as in "P is my reason for Q because I inferred Q from P." We corrupt the tongue to mean instead the inference step itself, so that if the program infers Q from P, not P but the record of that inference is called a reason for Q.

The importance of these reasons lies not in just the historical and explanatory information they provide, but in that the program uses the current set of reasons to determine the current set of actual attitudes. Thus some potential belief may have several reasons recorded for it, but if none of these reasons is *valid*, that is, refers back to current beliefs as antecedents, the belief in question will not be an actual, but merely a potential belief.

Reasons are recorded for all types of inferences, not just deductive inferences of one belief from other beliefs. Reasons record the inference of desires from other desires, intentions, and beliefs, and the inference of intentions from desires, beliefs, and values in decision-making.

An important property of these reasons is that they are *defeasible*. That is, after an inference has been made, it can be reflected on. If reflection determines that the reason was mistaken because, for example, the inference was made in exceptional or special-case circumstances in which it was not strictly valid, the program can defeat the reason by providing a defeating reason. This defeating reason may in turn be defeated by other reasons. The defeasibility of reasons allow the program to change any of its attitudes, for each attitude is held only because of some reason, and can be rejected by defeating all of its reasons.

The current sets of concepts, reasons, beliefs, desires, and intentions comprise the program's *current state of mind*. In fact, the program is a single concept "containing" all other concepts and attitudes, including itself. The procedures of the program are also concepts. Some of these make up the action-taking part of the program, called the *interpreter*, which reflects on the current state of mind and then acts on the basis of what it sees. (See Figure 2.)

The program often takes actions and inferences by executing *primitive procedures*, and it records these actions as statements. All primitive procedures are treated as attitudes as well, so when procedures make inferences, they record these actions as reasons, and include themselves in the reasons. Primitive procedures with external effects are recorded in somewhat different form, but that will be described later.

In addition to primitive procedures, the program embodies some of its skills in *plans*, which are concepts describing (roughly) patterns of desires and intentions. The program carries out its intentions either by executing a primitive program, or by reducing the intention to a plan, that is, by embracing or inferring the new desires and intentions specified by the plan. These plans and the desires and intentions they produce are reflected on by the program as a means towards controlling its actions. They form the self-conscious "tip of the iceberg" which controls the vast majority of computational steps taken unconsciously by primitives.

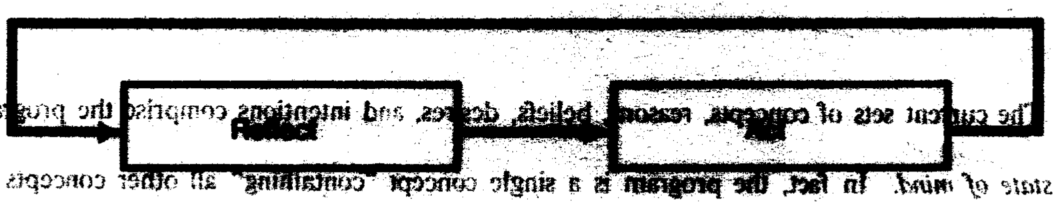


Figure 2

The Basic Cycle

The program's skills involve not only procedures but also statements about when these procedures are useful. Each *method statement* expresses that some procedure is relevant to carrying out some desire or intention.

Method statements about procedures and the aims of desires and intentions comprise just one special sort of information that the program may have about a procedure. More generally, the program employs statements of other properties of the procedure in other cases of reasoning. For example, one sort of property is that of input-output behavior. These are modal statements of the form "If P holds before the action, then Q holds after it." Other sorts of statements express properties concerning complexity or intermediate states of execution of the procedure. We will not often use or pursue such more general action properties in the following. However, one key type of information about plans is that of the relationships between plans. This information is expressed as statements relating one plan concept with another, such as that one plan is a refinement of another.

The program forms some intentions not by reducing an intention to a plan but by deliberation, by deciding what desire or intention to pursue or how to carry out some intention. These deliberations make use of *policies*. Policies are intentions which embody the values of the program, and are carried out by reasoning in decision-making. Policies are used in *reasoned deliberation* to indicate new options to consider and to give reasons for or against the options. Policies effect values by constructing reasons for and against other reasons so as to influence which option the program acts on by influencing which reasons are held to be valid grounds for action. The typical case of deliberation involves policies creating some options and conflicting reasons for what to do, and other policies reflecting on these reasons to apply the values of the program by defeating a lesser reason in terms of a stronger. These values are not expressed numerically, as is traditional, but rather as explicit statements that one particular sort of reason, in some particular set of circumstances, overrides some particular application of another reason. This approach to decision-making allows conflicting values to be settled or reconciled in a case by case manner, since the defeasibility of reasons means that any particular application of a value may be overridden if special circumstances so warrant. This approach also allows for the occurrence of dilemmas, for two types of values may be incomparable to the program.

Reasoned deliberation is used in many ways in the program. The most basic use is in deciding what to do next, in which the program reflects on which desire or intention to pursue and then on how to pursue it. But deliberation also guides the program's actions in other ways, the most important ways being deliberate changes in the program's set of concepts, beliefs, desires, intentions, values, and skills.

After making inferences, making observations, or taking actions, the program sometimes discovers a conflict between some of its beliefs. The normal path to follow in these cases is for it to discard some of its beliefs and assumptions to restore harmony. But belief revision always involves ambiguity, in that there are always many possible changes in the set of beliefs which will restore consistency. To decide which revision should be made, the program deliberates about the possible revisions and reasons for them. Formulating the possible revisions involves tracing through the reasons for the conflicting beliefs to find the underlying beliefs causing the conflict. The values of the program enter this deliberation by preferring one possible revision to another, effectively determining the tenacity with which the program clings to one set of beliefs rather than another. The program normally carries out this intention by defeating the justifications for the beliefs to be discarded and perhaps by justifying the opposite beliefs.

The program modifies its set of skills in a related way. If it determines that some skill does not live up to its intended specifications, the program will adopt an intention to decide how to modify the procedure, or the set of skills, so as to realize the intended specifications. To do this, the skill modification procedures employ deliberation to decide what sort of change is necessary, to decide what particular plan to fault for the problem, and to decide how to patch the plan to remove the problem. Determining what the possible changes in the set of skills are and how to make them is more complex than just examining existing reasons as sufficed in belief revision. Instead the program must often introspect into its primitive procedures to find the explanation of their behavior in terms of underlying plans. After it does this, it uses these plans in symbolically executing the primitive to see exactly how the problem occurs. It then analyzes the reasons for the problem in terms of the beliefs, intentions, and actions of the primitive in this symbolic execution to classify the problem into one of a number of problem types. It then deliberates on how to modify the procedure so as to avoid the problem. Once it has decided to make some particular modification, it modifies the plans involved in the procedure's construction, and compiles these plans back into the form of a procedure.

Skill modification plays a crucial role in the efficient operation of the program. For efficiency, most steps of most actions must be taken unconsciously, and skill modification techniques are the means for producing such unconscious skills from the prior conscious plans and experience with their use.

## 1.3 Outline of the Thesis

Chapter 2 describes SDL,[4] the language in which concepts and attitudes are phrased. Chapter 3 introduces RMS,[5] the underlying subsystem which implements the theory of reasoning. Chapter 4 describes the hierarchical library of plans and the interpreter, the action-describing and action-taking parts of the program. Chapter 5 explains how these techniques are combined in reasoned deliberation. Chapter 6 explores the application of these techniques in deliberate changes of the program's concepts, beliefs, desires, intentions, values, and skills. Chapter 7, the final chapter, discusses incompletenesses in this work, related directions for future research, and speculative topics.

[4] This acronym stands for *Structured Description Language*.

[5] RMS is a revised and renamed version of TMS [Doyle 1979]. The acronym stands for *Reason Maintenance System*. I am changing the name for two reasons. First, TMS, the *Truth Maintenance System*, has nothing to do with truth, and this misnomer has apparently annoyed some who took it more seriously than was intended. Second, as discussed in more detail in the last chapter, RMS maintains reasons for several sorts of attitudes, such as beliefs, desires, and intentions, so that it seemed prudent to name it after the reasons being recorded than after one of the attitudes (such as belief) being derived from these reasons.

## 1.4 Sketches of these Ideas in Practice

To illustrate how these sorts of techniques might be applied, we present several motivating sketches of reasoning in common situations involving decision-making, recollection, self-improvement, planning, and conversing.

### 1.4.1 Decision-making

Suppose that Robbie is a male robot. As Robbie is opening a closed door with the intention of walking through it, he detects an approaching object. He identifies the object as a woman (or perhaps a female-appearing robot), and considers what, if anything, to do about her. He thinks of two possible courses of action, (1) holding the door for the woman, and (2) ignoring her, thereby letting her open the door on her own. Robbie first forms a reason (a) for option (1), that chivalry demands a gentleman hold the door for a lady. Robbie continues to think and realizes that the modern woman finds chivalry an insult to her humanity, which constitutes a reason (b) against the first reason, that is, a reason not to act for reason (a). At this point Robbie still has no reason for action, since reason (a) has been defeated by reason (b), so he thinks further that he should hold the door by reason (c) of general politeness towards one's peers. At this point Robbie stops deliberating on what to do about the woman, and since option (1) has a good reason for it and (2) does not, Robbie decides to act on reason (c) and hold the door for the woman.[6]

Robbie next thinks about how he should go about holding the door for the woman. He considers the possibilities, the first of which is his standard method, that of (3) holding the door after he has passed through just long enough for the woman to reach the door and hold it herself. But he then

[6] If Robbie instead had been a time traveller to the early part of the twentieth century, he might have, unless he was very dull, realized that he had a reason (d) against reason (b), namely that the different time period made his original objection invalid. In this case, Robbie could have acted on reason (a) alone, for reason (d) being valid would make reason (b) invalid, thus allowing a good reason (a).

recognizes the woman as a friend whom he has not seen for some while, and considers the second possibility of (4) holding the door until she has passed through, following her through, and offering greetings as she passes. In this case, option (3) is his default door-holding method, and has a reason for it that is valid if there are no other options with good reasons for them. But Robbie also has the good reason of renewing a friendship for option (4), so the default reason for (3) is defeated. Thus Robbie decides on (4), holds the door, and says hello.

Although this example is informal, exactly the same techniques are important in highly constrained technical domains (not to imply that social behavior is not also highly constrained). For example, when writing a program one has a decision of how to implement some function. One possibility might be simple, another complex. One might have a reason for the first in its maintainability, but defeat that because of its inefficiency. One might defeat the reason of inefficiency because the program will receive only limited use. Then one might defeat the reason of maintainability because the simple method actually runs quickly on the cases of interest. Whatever the problem, one still has to somehow combine different sorts of values and exceptional cases in decision-making.

### 1.4.2 Recollection

We often would like the program to explain its actions, and normally it can do this by examining its records of its actions for the action in question, and then explaining the action in terms of the intention that led to the action, and then in terms of the beliefs, plans, and decisions that led to the formation of that intention. But what if the action in question cannot be found in the history of actions? The program then reasons about whether it took the action or not. It might possess information about its procedures (either through introspective analysis or design) sufficient to tell whether the action in question might have been taken unconsciously by some primitive. For example, the program might record its action of moving one block to a new location, but if the primitive it used to carry this out first moved some other block to clear the top of the target block without noting this subsidiary action, the program would miss this action in its history. However, if it knew that the block movement primitive could be invoked by primitives as well as through intentions, it could admit that it might have taken the action, but not consciously. If it knew even more about how the primitive might be called, the program might be able to infer that it must have been called and why it was called. The program might also try to recognize the action as emerging from the larger pattern of the actions it does recall. For example, the program might move some block around on a table until the block rests again in its original position. However, it might have to infer from its recollection of each of the separate actions that it took the action of leaving the block in place. Finally, the program might believe that the only way the action in question could have been taken was deliberately through an intention, and infer from this and the absence of any record of the action that it did not take the action.

### 1.4.3 Self-improvement

> How pleasant it is, at the end of the day, no follies to have to repent;
> But reflect on the past, and be able to say, that my time has been properly spent.
> Jane Taylor, *Rhymes for the Nursery. The Way to be Happy.*

In addition to reflecting on its actions to explain them, as in the previous sketch, the program might also reflect on its recent actions to see if they signal any changes that should be made in the procedures used in taking these actions. I, for example, reflect on the day's events each night before going to sleep. I also reflect on recent actions when I get annoyed with something, to see if I can think of some way of avoiding similar annoyances in the future. In recent times I recall several discoveries I made in this way which I then put to use in improving my future performance. For example, I used to shave after showering. Having to wash my face after shaving eventually annoyed me enough so that I realized that I wouldn't have to wash my face a second time if I shaved prior to showering. So I switched my routine. However, I later became annoyed with the stiffness of my beard, which on reflection I attributed to the lack of shower softening, so I switched back, now the wiser about toilet techniques. In the same way, an intelligent program might be fruitfully organized to reflect on the efficiency of its past actions both when problems arise, and as a regular matter (once per day as the nursery rhyme goes, or during conversationally idle periods).

### 1.4.4 Planning

Regular review of one's plans often results in their modification, for example, by realizing their incoherence, their inappropriateness, or their importance. For example, the program might decide to carry out two of its intentions by means of plans. Unless it then reflects on these plans, it might never discover that together the plans have substeps calling for simultaneously unrealizable or needlessly repetitive actions. The program can correct these problems by carefully ordering the steps, or by discarding one, or by inserting new steps to mitigate the interference between the separate plans.[7] The program might also notice that a great many of its intentions turn on some decision it intends to make. In this case the program might explicitly state the importance of the decision, and adopt the intention to be very careful in making the decision, that is, to use a careful deliberation procedure rather than to decide quickly.

### 1.4.5 Conversation

In addition to reasoning about its own actions and attitudes, engaging in conversations requires that the program reason about the actions and attitudes of others as well. Intentions to inform can be analyzed as intentions to have the other participants in a conversation believe some fact. Intentions to request something of someone can be analyzed as intentions to inform that person that one has a certain desire

[7] Sacerdoti's [1977] NOAH is an example of ways in which such reflection and action might be done.

whose satisfaction involves their cooperation. Furthermore, an intention to persuade can be analyzed as an intention that the other person adopt a certain desire.

In all these cases, to plan one's utterances one needs to reason not only about one's own attitudes and actions, but also about the other person's attitudes, his attitudes about one' own attitudes, and the beliefs and skills in common to both participants in the conversation.[8]

To perform this sort of reasoning, the program might make copies of its own mental structure, interpreter, library of procedures, etc. to represent each other participant, and then simulate and interrogate these models to predict what the effects of its own conversational actions will be.

### 1.5 Status of the Implementation

> Can these bones live?
> Ezekiel, xviii:27

No complete, working, fully tested version of the program exists at the present writing. This section explains both what has been implemented, and forseeable difficulties in completing the implementation. All of the parts I have implemented are written in LISP for the MIT Lisp Machines.

Several versions of many parts of the program have been implemented and experimented with to varying degrees by various people. SDL is based on a modest extension of the ideas used in FOL [Weyhrauch 1978]. An implementation of FOL by Weyhrauch and others has been working for some time and applied to several projects. SDL has been implemented several times, but never as completely as its description in Chapter 2 indicates. RMS is a modification of TMS [Doyle 1979]. TMS has been used extensively in many programs. RMS itself has not been fully implemented or tested. The interpreter is an extension of the "task network" interpreter used in NASL [McDermott 1978]. NASL is a

[8] Speech-act approaches to discourse have been attracting increasing attention recently. For background and current proposals, see [Austin 1962], [Searle 1969], [Grice 1969], [Cohen 1978], [Grosz 1979], [Perrault, et al. 1978], and [Wilks and Bien 1979].

working, tested program. Charniak has recently reimplemented a subset of NASL as well. My interpreter has gone through several versions, each of which was tested on small problems, although none of these versions has all the complexity of the one described in Chapter 4. Similarly, the deliberation techniques in Chapter 5 have received an initial implementation and testing through their use in the interpreter. Some of the techniques of Chapter 6 have been tested, others are completely untested, and still others form the content of other works, such as those of Winston [1975], Sacerdoti [1977], and Sussman [1975].

The major reasons for the lack of a complete implementation are three: a lack of time on my part, my confusion about how to implement databases, and inadequate computing resources. This thesis synthesizes a large number of ideas, making it impossible to treat them in greater detail within a reasonable period.

Hierarchical databases, of the sort used in SDL, have received considerable attention by many authors, and many implementations exist. However, I had none of these readily available to me, had my own peculiar requirements for extensions to them, and continually procrastinated on the task of reimplementing one for my own use. There are many subtle problems involved in the exact details of these databases, and although I have substantial interests in these questions, they were not the questions I wished to pursue in this thesis, so I exerted little effort on resolving them. The basic ideas of Chapter 2 I have known for some while, and have taken much of the actual detail of the structure of theories directly from Weyhrauch's system.

Straightforward techniques for implementing reasoning programs along the lines described above require a substantial overhead in time, space, and notation. At first glance, the techniques require recording semi-permanently many sorts of information that traditional programs either never consider or only record very sketchily and then discard quickly. This increases the constant factors of the complexity of the program on the order of 100 times over the space requirements of traditional programs. (100 is just an off-hand, possibly pessimistic guess, and depends on the implementation techniques used.)

I have not been overly concerned with this overhead, for a key point of my methodology has been that it is too expensive *not* to record and use this information. I repeat: It is too expensive *not* to record and use this information.[9] The standard approaches suffer unavoidable combinatorial explosions in searching because they discard the very information that might be used in bypassing these fruitless searches. I accept large increases in the constant factors to gain the ability to kill the exponential terms of the program's complexity, and to instead achieve a program complexity which grows roughly linearly with the complexity of the problem. The issue is not my skill at programming. Instead, the issue is to analyze what information is necessary or at least useful in steering the program clear of these searches, and then to develop ways of recording and using this information.[10] I concentrate on the asymptotic complexity of the techniques involved, on the fundamental concepts involved in control. This is important, for it means that as the problems become larger and more complex, a linear time program remains feasible even if its constant factors are very large, whereas an exponentially expensive program is always useless, no matter how efficient it was on small problems. Combinatorial searches cannot be the basis for intelligence. They will never be fast enough. The problems always get harder to quickly.

The unfortunate consequence of attacking the fundamental problems of reasoning is that current computers are too slow and too small to permit debugging of programs. It is nearly impossible to make progress debugging a program which takes several hours of interactive operation to manifest each new error and which must be started from scratch after each patch (as programs under initial development require). However, this is just what happens. I have written programs to solve unremarkable problems that represent exactly the information that seems necessary, that make only the inferences which must be made (i.e. no wasted searches), but which on absolutely trivial problem

[9] I here repeat a statement made by G. J. Sussman [Latombe 1978, p. 364].

[10] For example, the technique of dependency-directed backtracking developed by Stallman and Sussman [1977] was an effort to use a fixed overhead of extra records of dependence of results on assumptions to avoid the needless combinatorial searches required by traditional chronological backtracking. A similar motivation gave rise to the separation of database and control information in CONNIVER [Sussman and McDermott 1972].

instances spend a quarter-hour of CPU time (and hours of real time) exhausting the address space of MIT's DEC KA-10. MIT Lisp Machines provide a faster interpreter and a larger address space, but quickly become disk-bound, and then spend most of their time paging, just like the KA-10.

The key factors in the debuggability of these programs is the speed of the machine, and the size of real memory. The sort of program described in this thesis can, I expect, reasonably be implemented and tested only on machines a thousand times larger (and perhaps faster) than the computers mentioned above. Such machines may exist affordably within the next decade, and we must forego hope of true intelligent machines until then.

What can be done meanwhile? I believe we should work on problems towards that day when suitable machines exist. It is not enough to concentrate only on problems whose techniques can be implemented on current computers. For many important problems, those techniques are sure to be unsatisfactory, substituting searches for intelligence. Science progresses not by building programs which initially run "efficiently" but cannot in principle run fast enough, but rather by building programs which are feasible in principle, even if we must build new computers to run them. Imagine the result if Beethoven had tried to compose his Ninth Symphony for solo voice and pianoforte. I have no doubt he would have produced something, but it wouldn't have been the Ninth Symphony, and would not have "solved the problem" or said the same thing that the Ninth Symphony did.

### 1.6 Sketch of a Computational Argument for the Approach

> All reformers are bachelors.
> George Moore, *The Bending of the Bough*

The standard view in AI research has been quite different from the conscious, adaptive, reasoning approach outlined above. This section hypothesizes a strawman view to stand for the traditional AI approaches, and speculates on how it came to be adopted. It then attempts to give some insight into the computational motivations for the proposed approach by means of computational criticisms of this strawman view. This argument is made indirect for two reasons. First, those familiar with the traditional approach will see the main limitations of that approach. Second, these criticisms will suggest how realizations of those limitations might lead to the view proposed here.

### 1.6.1 Why have the facts of the fundamental argument been overlooked?

It seems clear that most AI research misses the above ideas completely. Judging from almost any volume of conference proceedings or journal issue, one sees the overwhelming emphasis on either designing a black box algorithm for some problem, or for designing a formal language for writing down the axiomatization of some particular domain or class of domains.[11] Of course, such studies are often necessary precursors of continued progress, but the question remains of why control, consciousness, and adaptiveness have received so little attention. The following subsections suggest two possible answers to this question.

#### 1.6.1.1 Initial Programming Complexity

The simplest answer is the large overhead required by the techniques described here, and the consequential undebuggability of programs based on these techniques. This makes problems admitting more immediately testable solutions more attractive in some ways.

[11] See [Brachman and Smith 1980, p. 3], who conclude that "far more people claim to represent the world than claim to represent knowledge."

#### 1.6.1.2 A Mathematician's Outlook

I think a deeper reason why the fundamental argument above has been overlooked has to do with the way of thinking implicit in the traditional approach. The standard view seems characterized by an obliviousness to change, a blindness to the need for the program to continually adapt itself to changing environments, tasks, and patterns of use. AI has tended to view the problem of representing information about the world as that of defining several basically fixed (logical) theories, and using a single basically fixed set of programs for reasoning about these representations. If new theories or reasoning procedures are required, the AI researcher writes a new program, rather than helping the program to change its old ones. Perhaps I am being unfair to mathematicians, but this seems to result from sharing the typical mathematician's outlook on knowledge. Mathematicians discover concepts, theories, and theorems, but once they have given a name to something, they never consciously change the meaning of that name. If they discover that the named concept was not quite the interesting one, they make a new name for the new concept, rather than changing the meaning of the old one, so that mathematical theories are impervious to change. Since mathematicians do not often explicitly concern themselves with the use of their theories in their studies, they are also somewhat blind to changes in how these theories are used in reasoning. A book on determinants written today would likely have the same form and theorems as one one written when the subject was alive.[12]

AI tends to formalize a theory of blocks, natural numbers, or elephants, and once this axiomatization is set, it is rarely changed. Instead, modifications are given new names. AI adopts a standardized form for reasoning, say resolution, production rules, procedural attachment, or what have you, then lets this organization sit untouched in its reign over all domains. Since the basic representations and reasoning processes are fixed, the AI researcher can build them into a program, and, to improve the

[12] Here I am deliberately exaggerating the point for the sake of argument. Dead fields sometimes regain popularity through the infusion of new methods from other areas, and Dummett [1973] and Lakatos [1976] might be taken as suggesting that mathematicians unconsciously change the meanings of their terms.

program's initial efficiency, discard most of the information concerned with why these representations and processes are used. But these reasons for the current organization are just what is necessary for the program to be able to reason about how to change its organization when its environment or usage changes. Blindness to change leads to organizing programs so that their representations and reasoning processes are built-in, unchangeable. This is the traditional view's fatal flaw.

### 1.6.2 Consequences of the Inaccessibility of Control Information

> Sing, O Goddess, the anger of Achilles, son of Peleus,
> that brought countless ills upon the Acheans.
> Homer, *The Iliad*, translation by Samuel Butler.

We label the traditional view's fatal flaw the *inaccessibility of control information.* Just as the inaccessibility of captain Achilles contributed to the Acheans' woes at Ilium, the singularly unhappy methodology of the inaccessibility of control information leads to manifold unhappy consequences. Since there is just one correct way of organizing reasoning, the framework-systems investigated in AI usually support only one program at a time. The researcher has the responsibility for determining what that program should be and for coding it up. He is also responsible for writing a new program or changing the old one when the program is discovered to be in error or inadequate to its task. That is, the program is not organized to be adaptive, but the programmer is expected to do the adapting.[13] For example, almost all the early programs (such as SHRDLU) written in PLANNER[14] required that all changes be made by the programmer.

This non-adaptiveness has a terrible consequence in practice. Because all responsibility for the

[13] A frequent symptom of both this problem and a limited control vocabulary (Section 1.6.2.2) is the oft repeated warning of system designers that the *user* should take care in deciding which inference rules should be used for forward chaining and which for backwards chaining. This is a good signal that something is wrong with the system.

[14] I will give most of the examples of traditional systems and their problems in terms of PLANNER, in part because it so clearly demonstrates most of these problems, and in part because so many subsequent systems are largely based on its ideas. The full language introduced in [Hewitt 1972]. However, only a subset was ever implemented, and the examples refer to programs written in that subset. The full language shares all the problems of the subset.

writing, maintenance, and evolution of the program is kept by the researcher and none is given to the system, the information describing the program and its organization is typically distinct from the information with which the program reasons. For example, references to PLANNER's control stack had to be made in LISP, rather than in terms of PLANNER assertions and theorems. Almost always, the program cannot refer to its own structure and the structure of its behavior as its designer does.[15] Since this is information controlling the program's reasoning and actions, we call this the inaccessibility of control information. The program simply cannot reason about its own control processes.

In the following subsections we outline some of the many unfortunate consequences of the inaccessibility of control information. The nesting of subsection numbers will reflect the consequential relationships between these difficulties.

#### 1.6.2.1 The Inexplicability of Actions and Attitudes

The first problem following from the inaccessibility of control information is the *inexplicability of actions and attitudes*. Because the program cannot interrogate its own control process, it cannot explain why it took the actions it did, why it didn't take the actions it didn't, why it believes what it does, and why it plans to do what it does. For example, early evaluation-function search techniques rarely kept records of their searching actions. Instead, they were notorious for basing all actions on inexplicable and uninformative numbers.

One might think that this inexplicability is a trivial flaw, that one can tolerate incomprehensible programs. This, however, is an gravely misguided tolerance. As programs and databases become more common and more complex, society comes to rely crucially on their accuracy and intelligibility. Stories abound of false information irrevocably ruining someone's credit ratings, employment records, or worse.

[15] By this is meant the terms and reasons with which the designer explains the program's design. These explanations include much more than just the programming language in which the program is written, at least with current programming languages.

In trying to deal with such tragedies, society finds that computer systems are designed with the view that they are monolithic, infallible sources of information. This leads to great disrespect and growing resentment of these large information systems. If we are to justify our reliance on these systems while avoiding society's censure, we can take either one of two paths. We might make programs responsible for their actions or, more immediately practical, we can make programs explicitly defer all responsibility to humans. We can have programs keep historical information about their inputs and about the computations they perform. This historical information can then be used to construct explanations or justifications of each action and database entry so that errors can be traced to bad inputs, to faulty programs, or to other databases in a distributed system. In this way, the computer can be prepared with the fact of its own fallibility and irresponsibility, and can help track down its own problems and those of its users. While this may not render intelligible the enormous systems of programs involved, at least their effects will have been isolated to some extent. It may be impossible for programs without historical annotations to do many of the things that we want them to do, namely to defer responsibility to humans so that their actions may be explained and corrected. The larger and more important programs become, the more important such humility becomes. The fairness and effectiveness of programs are at stake, and if society is to trust their accuracy and usefulness, they must be able to trace their actions and contents to responsible sources.

In addition to these strong social reasons against incomprehensible programs,[16] many important limitations on the program stem from the lack of reasons for actions and attitudes.

[16] See also [Weizenbaum 1976] and [Rosenberg 1980].

#### 1.6.2.1.1 The Chauvinism of Values

> But are they all horrid, are you sure they are all horrid?
> Jane Austen, *Northanger Abbey*

The most important problem stemming from this inexplicability involves the *chauvinism of values.* The inability to examine one's reasons greatly limits the sorts of decision-making that can be performed, for it forces one to fit all sorts of values into a single dimension, thereby making impossible reasoning based on the incomparability of values.

Dilemmas are the central problem. The genesis of dilemmas is in part that we think of our world comprising many subworlds, with only tenuous connections between them. We can describe the world and our actions in physical terms, or from the standpoint of a moral system, or as events in an economic system, or simply in terms of what we like and dislike. Each of these subworlds of physics, morality, economics, or pleasure has its own vocabulary, facts, principles, and values. The values of each of these systems cannot be compared with the values of the other systems. If we eventually discover some reduction of all these worlds to a single world, for example, some way of reducing moral and economic theory to physics, then we may have hope of comparing a moral value with an economic value. Without such a revolution, however, we must live with incompatible values. Indeed, many thinkers have argued that we will never find such a reduction of values because one does not exist, or that even if one did exist, the explanations for decisions resulting from the reductions would be too detailed and intractably long for routine purposes.[17] We must, at least for the time being, find some way of making decisions despite this *fragmentation of values.*[18]

This fragmentation of values permeates our deliberations far more than one might expect. Even in apparently technical decisions, which in the popular view are the most straightforward, incomparable

[17] See Fodor [1975] and Putnam [1975].

[18] This is Nagel's term [Nagel 1979b]. Bell [1976] terms it the "disjunction of realms".

principles must be reconciled. For example, when designing an automobile, or a computer program, or an electronic circuit, one typically encounters many decisions between different ways of implementing the design specifications. But when one comes to these decisions, one must choose between methods which result in varying degrees of elegance, expense, ecological harmfulness, reliability, ease of maintenance, conformity with statutes, coverage under patents or patentability, difficulty of design, complexity or size of the design, ease of construction, ease of customization, the favor of one's peers, the innovativeness or personal challenge of the design, marketability, workability under expected future changes in energy, legal, and social systems, etc., etc., etc. All of these considerations involve different sets of values, and in anything one would call a problem, the value of the decision cannot be maximized along all of these dimensions simultaneously. Making decisions necessarily involves reflecting on the types of reasons involved to compare them with each other. If these reasons are incomparable, then the decision cannot be made in a fully rational fashion when so desired.

The fragmentation of values is a strong motivation for avoiding systems which do not record their reasons, or which use only reason-obscuring techniques like voting or numerical strength-combination rules for decision-making. In this latter case, such systems impose arbitrary, implicit, and frequently indefensible judgements about the relation of different types of reasons by chauvinistically fitting all types of reasons into a single-dimensional grading scheme. For example, MYCIN [Davis 1976] forces all decision-making into numerical strength-combination rules. This not only means that the program must commit itself to absolute strengths for all reasons, but it also means that the combination of reasons cannot be affected by context. A classic instance of this is the intransitivity of evidential relationships in medical diagnosis. As Rubin [1975] explains (along with other examples), both facial edema and ascites are evidence for sodium retention, and sodium retention is evidence for each of cirrhosis and acute glomerulonephritis. However, these evidential relationships are not transitive, as would be required by MYCIN, since facial edema is always positive evidence against cirrhosis, and ascites is positive evidence against acute glomerulonephritis. Here context (i.e. facial edema) invalidates a usual evidential relationship (i.e. between sodium retention and cirrhosis). Even Simon's satisficing decision-making, which avoids the unnatural *homo economicus* or value-maximizing man, still fits all utilities into a single dimension [Simon 1976]. Necessarily chauvinistic decision-making processes may be simple, but lead to insurmountable inadequacies in the reasoner, and lead to more decisions being made than is properly possible.[19]

### 1.6.2.1.2 The Lack of Intentionality

Another problem stemming from the inexplicability of actions is the *lack of intentionality*. If the program cannot reflect on why some action was taken, or why some circumstance occurred, it cannot distinguish between the intentional and the unintentional consequences of an action. A famous problem with PLANNER-based robot bank robbers is that they would blithely proceed to rob the bank after tripping over a pot of gold while on the way to the bank. Being able to make these judgements is crucial in analyzing its successes and mistakes with an eye to improving its skills and performance. Telling whether the effects of some action were "successful" or not depends on the ability to distinguish some conditions as the aim towards which the action was taken, and then checking if the action realized these conditions. For example, I have on occasion begun to assemble a complex toy without understanding what the intended structure was. When the assembly directions were unhelpful and did not explain the intended functions of the parts to guide me, I sometimes completed the bulk of the assembly only to find that I apparently misassembled some substructure earlier because the next assembly instruction made no sense for the then current partial assemblage. To correct my error, I tried to reconstruct the intentions of each assembly step and see where my actions had diverged from the intended actions.

[19] Of course, any decision-making procedure may be made chauvinistic by a decision to accept universal comparison rules. The issue here is whether the decision-making procedure forces this decision on one, or whether one can leave some values incomparable.

#### 1.6.2.1.3 Inextensibility

The inexplicability of the program also contributes to the *inextensibility* of the program. Since the program cannot explain its workings, it has little chance to aid in its own modification. Even trivial changes must be left to the designer or user to effect. Simple methods for augmenting the procedures used by the program, such as those presented in [Davis 1976], are impossible to implement. For example, the program may in the course of reasoning discover that its beliefs are inconsistent. If the program can explain its beliefs, it can help to trace the conflict back to its assumptions and to resolve the conflict by changing one of these assumptions. But without a self-explanatory facility, the program's extender must rely on the program's designer to provide this analysis, if he can. For example, PLANNER recorded no explanatory information outside of its control stack, and that was sufficient only for suspecting the chronologically last procedure executed, as in chronological backtracking.

#### 1.6.2.1.4 Hubris

The fifth consequence of the inaccessibility of control information and the inexplicability of the program is its *hubris*, the program's inability to acknowledge its own fallibility and limitations. Because its reasoning and deliberations are external to its language, it cannot say anything about whether it might be wrong in making some inference or decision, but has to proceed as though it is always right. In fact, the program has many limitations in its abilities and in its knowledge of its abilities. Its knowledge of its own abilities and beliefs is not very much more secure than its knowledge of the external world. To be effective in action and in reaction to difficulties, we must replace *hubris* with *sophrosyne*, knowledge of both abilities and limitations.[20]

Many useful forms of reasoning depend on being able to refer to such limitations. A prime

[20] For a better explanation of this term, see Ostwald's gloss of *sophrosyne* in [Aristotle 1962, p. 314] and Aristotle's usage in Book 3, Section 10 of that work, pp. 77ff.

example of this is the ability to make default assumptions or other non-monotonic inferences on the basis of incomplete information. Such inferences can be made and maintained correctly only if the reasons for beliefs and actions can be given. This was one of the major failings of PLANNER and its relatives. PLANNER could not correctly handle THNOT because it lacked reasons for its beliefs. PLANNER could not correctly compute the intended conditions of a THNOT's success, for it could not tell which assertions depended on previous THNOTs. For the same reason, it could not correctly update its set of assertions when a new assertion invalidated a previous THNOT's success. Knowledge of one's limitations also enters into the tenacity with which one holds beliefs, into judgements about which beliefs to give up (say as tentative hypotheses) before others (say as tenets of faith).

### 1.6.2.1.5 Non-additivity

A fourth result of the inexplicability of program actions and attitudes is a *failure of additivity.* This problem involves more than non-monotonic changes in the program beliefs in response to actions and inconsistencies. Here I refer to PLANNER's failure under the addition of new imperative inference rules. The programmer always had to take great care when adding new inference rules to avoid loops of inferences which would halt progress. Not only would some added rules cause catastrophic failures of the program through non-terminating iterations, but no information could be added later to indicate the proper use of the rules. For one example, a backwards-chaining inference rule, to the effect that one block is a above another if there is a block which is above the one and below the other, might never halt if asked about two isolated blocks. If asked to prove that block A is on block B, it would generate the subgoal of finding a block C above B and below A, the sub-subgoal of finding a block D above C and below A, and so on, endlessly, without the possibility of adding a new rule to say that the first rule should never be used if the two blocks are isolated. Also, conflicting non-monotonic rules will loop. If one has procedures (each added by a different user with his own ideas about what the program should do) to add

P whenever Q is added, to erase Q whenever P is added, to erase P whenever Q is erased, and to add Q whenever P is erased, one might go into an infinite loop of adding Q, adding P, erasing Q, erasing P, adding Q, etc. No techniques fully adequate to this problem were available because the real answer involves keeping track of the inferences themselves, that is, the actions of the control component, and reasoning about the presence of loops in these inference records. When such techniques are employed, new inference rules may be added without fear of this sort of failure occurring.[21]

### 1.6.2.2 Inexpressibility of Control Information

The second major problem stemming from the inaccessibility of control information involves the *inexpressibility of control information*, the inability to give the program heuristic advice, guidelines for how to carry out an decision or task. This is the doom of McCarthy's goal of an Advice Taker [McCarthy 1958]. Because control is fixed external to the program, at some point the controller must arbitrarily give up on controlling the program's actions and resort to blind search, for otherwise the control component would contain all possible information about how to do what, when. For example, the usual PLANNER scheme of writing programs with inference rules marked as forward or backward chaining allows one to significantly direct the behavior of the programs in simple cases. However, when one increases the number of inference rules beyond trivial proportions, one finds many goals or assertions being answered by an unmanageable number of inference rules. It requires a new language of control to specify even the simplest procedures for directing what to do in this case, such as which rules are to be dealt with first. Some systems employ such a rule by imposing a linear ordering on the order of execution of all inference rules. Whenever one builds in a level beyond which the program can never see, one builds in eventual search, for any fact may at some time be the point on which an enormous search turns.

[21] For example, AMORD avoided these problems in just this way. [de Kleer, et al. 1977]

### 1.6.3 Hence Reasoning Applied to Control

The common element of all the above inadequacies of the traditional approach to reasoning programs is the inability of the program to refer to, to reason about, and to modify the information controlling its actions. The obvious approach to remedying these inadequacies is to design reasoning programs which can reason about themselves. In this way we can simultaneously overcome the limitations of previous approaches and make use of their strengths, for nothing need inhibit the program from consciously deciding to use one of the less sophisticated methods in certain cases if it deems those methods appropriate and more efficient in those cases.

## 1.7 Relation to Other Works

> And one might therefore say of me that in this book I have only made up a bunch of other people's flowers, and that of my own I have only provided the string that ties them together.
> Michel E. Montaigne, *Essais*

This thesis is related in general and in detail to a number of other works. Some of these will be cited in the chapters that follow. In this final section of this chapter, we first relate the thesis to its closest relatives among those works which have had the strongest influence on it. (See Figures 3 and 4 for a "mythical" summary of these influences.") We then survey some of the many other works relevant to topics studied in the thesis.

### 1.7.1 Major Influences and History

This thesis is an outgrowth of my earlier research on control of reasoning and belief revision. This line of work started for me with my paper "The use of dependencies in the control of reasoning" [Doyle 1976], which emphasized the need to control reasoning and "reasoning about reasoning" as a promising approach towards solving it. There I describe an early version of RMS, along with its application to

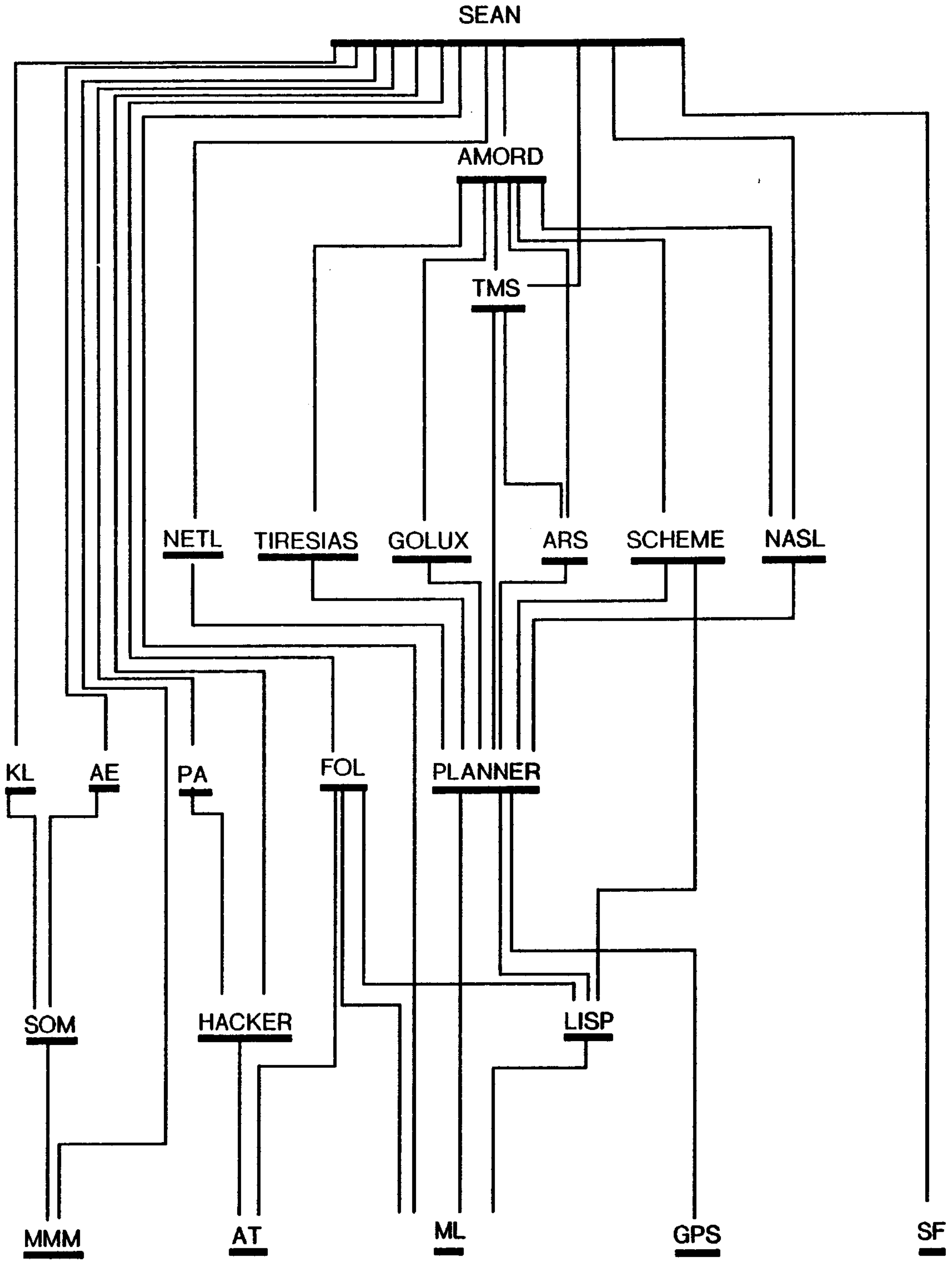


Figure 3

Diagram of Mythical Influences

(See next page for mnemonic interpretations.)

SEAN = this program
AMORD = de Kleer, Doyle, Steele, and Sussman [1977]
TMS = Doyle [1979]
NETL = Fahlman [1979]
TIRESIAS = Davis [1976]
GOLUX = Hayes [1974]
ARS = Stallman and Sussman [1977]
SCHEME = Steele and Sussman [1978]
NASL = McDermott [1978]
KL = Minsky's K-lines [1979]
AE = Minsky's Affective Exploitation [1980]
PA = Rich, Shrobe, and Water's Programmer's Apprentice [1979]
FOL = Weyhrauch [1978]
PLANNER = Hewitt [1972]
SOM = Minsky and Papert's Society of Mind [1978]
HACKER = Sussman [1975]
LISP = McCarthy, et al. [1965]
MMM = Minsky's Matter, Mind, and Models [1965]
AT = McCarthy's Advice Taker [1958]
ML = Mathematical Logic
GPS = Newell and Simon [1963]
SF = Asimov [1950, 1964] and Heinlein [1966]

Figure 4

Key to Influence Diagram Abbreviations

maintaining explicit statements of the goals of the reasoner. My masters thesis, revised as "A truth maintenance system" [Doyle 1979], developed RMS further along with its philosophy and applications. Developing the other theme of my first paper, "Explicit control of reasoning" [de Kleer, et al. 1977], proposed the explicit representation of the control state of the reasoner, in the main clarifying my earlier paper. This paper introduced AMORD, a procedural deduction system based on RMS, first implemented by de Kleer and Sussman.[22] In unpublished work, I extended the example system of this paper, and later Shrobe [1979b] extended it yet further. I hope that the present thesis ties these threads of thought together again.

I owe my colleagues large debts for many ideas. My earliest exposures to these sorts of ideas were, I believe, in a class on religion with Lloyd Swenson, and later, in my studies of mathematics with Joseph A. Schatz, who tutored me in the possibility of scrutinizing one's beliefs and rules of reasoning, and how such scrutiny is essential in foundational questions. John S. MacNerney vividly illustrated this point to me in a class on integration.

At the time of writing of my first paper above, I had been working for Sussman on ARS [Stallman and Sussman 1977] and with McDermott on NASL [McDermott 1978], and was very excited by their programs, and by Davis' new thesis [Davis 1976] as well. I then sat in on some discussions involving them, de Kleer, and Steele, thinking about the structure of a "new MICRO-PLANNER" based on antecedent reasoning. I then developed and applied my idea of non-monotonic data-dependencies to try to make some of these things workable, and my first paper above is the result.

RMS itself stems from my experience with the "fact garbage collector" of ARS. I introduced the idea of non-monotonic justifications for beliefs (and how they fit into dependency-directed backtracking) to capture the "PRESUMABLY" inferences in NASL. I discovered that ARS's fact garbage collector and backtracker were both very buggy and needlessly non-incremental, and isolated

[22] AMORD is actually the second program of that name, Steele and I having labored over and, after several months, finally quietly buried the first AMORD.

improved versions of these subprograms based on non-monotonic justifications as a domain-independent subsystem.

My interpreter is an extension of NASL's task network interpreter. NASL in turn builds to some extent on Sacerdoti's NOAH [Sacerdoti 1977], which reasoned about its own system of intentions represented as a "procedural net." My major changes to NASL have been the reorganization required by the RMS, the use of a hierarchical library of plans (NASL used the first order predicate calculus in such a way as to make this inconvenient at best), the separation of desires and intentions, and the introduction of reasoned deliberation. NASL's choice protocol is a simple relative of reasoned deliberation, with little of the structure, power, or intutiveness of the latter. In NASL's choice protocol, one erases options, retains options, or combines options, until just one option remains. One cannot give reasons against reasons, since there are no reasons. However, one can draw some conclusions about the deliberation process as a whole through the QUIESCENCE step of the choice protocol, which signals the executive that decision-making has gotten stuck. McDermott used this last ability for encoding default decision outcomes. NASL is little concerned with self-models, and so lacks plans describing the interpreter's actions. However, some of NASL's plan-reformulation mechanisms hint at a self-model, as they are mediated through plans rather than as simple procedures. .

The formalism for desires, intentions, and plans used in the interpreter is also related to the plan formalism of Rich and Shrobe [1976], who in turn refine earlier formulations [Brown 1976, Sussman 1975, Goldstein 1975]. They also present libraries of standard plans for programming, and methods for analyzing programs into their underlying plans. I draw heavily on their work in my approach to skill introspection and hypothetical reasoning.

Patrick Hayes has long advocated the general approach of controlling reasoning by reasoning about control [Hayes 1974]. He first suggested this idea in elucidating the relation between computation and deduction [Hayes 1973b]. More recently, he critiqued the traditional approaches to control [Hayes 1977a], and I have tried to build on his criticisms in my arguments above.

Long ago, McCarthy proposed an Advice Taker, a program which could accept facts and heuristics about the world and how to reason, and then find ways of using this information effectively [McCarthy 1958]. His proposal had no direct influence on me, but had a great indirect effect through Sussman's thesis [Sussman 1975], which has had a large impact on my views of learning of procedures and assimilation of information.

I have also been considerably stimulated by Sussman's addiction to writing meta-circular LISP interpreters [Steele and Sussman 1978b]. Although these interpreters do not refer to their self-description to act, they admit a description of themselves in the same language that they interpret, so that they can be used to evaluate themselves evaluating some other program.

I toyed with ideas about how to make a non-monotonic, hierarchical calculus of descriptions ever since reading about NETL [Fahlman 1979], which substantially influenced my views on databases. However, these ideas never demanded quite enough of my attention to permit their full development. My confusions about this might have hindered this thesis even more than they have, had it not been for Weyhrauch's timely exposition of FOL [Weyhrauch 1978]. I finally worked out the details of SDL while trying to understand his paper and its relation to Brown's work on meaning and meta-theory [Brown 1977, 1979]. SDL draws heavily on both NETL and FOL.

Minsky's ideas on reflection [Minsky 1965], once I discovered them, served to illuminate the problems I was fumbling towards. I was also extremely stimulated by his views on the role of affect in intellect [Minsky 1979], and by his criticism of the logistic approach in reasoning in [Minsky 1974].

Finally, many years ago I read and reread a number of stories which have ever since inspired my attitudes towards the problems of building intelligent machines. I would like to thank the authors of these stories, Isaac Asimov [1950, 1964] and Robert A. Heinlein [1966], for their inspiration.

### 1.7.2 Related Works

There are many other related works, some of which influenced me, but most of which were developed independently. Unfortunately, I am not quite the scholar I wish I were, and much of my recent and continuing effort has been directed to learning of the approaches already developed to the problems of this thesis and trying to relate all these ideas. However, I am still a novice in most of these areas. I have explored enough to see the truly huge bulk of writings on these topics, so to add some measure of coherence, I discuss them by topic.

#### 1.7.2.1 Representation Theory

This category groups together studies of the nature of representation, hierarchical representation systems, and self-descriptive and self-referential systems.

The *philosophy of logic and language* is the usual location for studies of the nature of representation, meaning, and representational system. Quine [1970], Haack [1978], and Linsky [1977] are good survey expositions of this area. Linsky [1971], Schwartz [1977], and Strawson [1967] are useful collections of articles on these topics.

Representational systems based on ideas of *hierarchical relations* between representations have been explored by Brachman [1978], Fahlman [1979], Minsky [1974], Ph. Hayes [1977], Hendrix [1975], Steele and Sussman [1978c], Smith [1978], Martin [1979], and Borning [1979]. Simon [1969] stresses the importance of hierarchical systems in organizing information and behavior.

There has been quite a bit of work on *self-descriptive* and *self-referential* systems, although most of it is foundational in character and little is applied to the problem of controlling reasoning. Programmers will find familiar the idea of the meta-circular interpreter, the earliest of which is Turing's universal machine [Turing 1936]. Such interpreters have also been developed for studying the semantics of programming and logical languages by McCarthy [1965], Backus [1973], Reynolds [1972], Brown

[1977], and Steele and Sussman [1978b]. Minsky [1965] discussed machines which reason about and use their own self-descriptions. So also do those mentioned above involved in the approach to control pursued in this thesis. Davis [1976] not only explores controlling reasoning with self-reference, but also shows the high value of programs using models of their own data structures and inference rules in acquiring new information.

In addition to meta-circular interpreters, the computer-architecture and compiler-compiler fields have studied formal machine description systems. See [Bell and Newell 1971], [Cattell 1978], and [McKeeman et al. 1970].

The fundamental formal properties of self-descriptive and self-referential systems have been studied by Russell [1908], Hilbert [1925], Godel [1931], Tarksi [1944], Turing [1936], Post [1943], Kleene [1950], Smullyan [1957], Montague [1963], Quine [1966], Kripke [1975], Feferman [1960], Resnik [1974], Boolos [1979], and Scott [1973]. Smullyan [1978, 1980] and Hofstader [1979] present popular expositions of some of these questions. The non-monotonic logics mentioned below can also be viewed as self-referential systems. Brown [1979] and Weyhrauch [1978] have each developed programs which can reason about languages, proofs, and models. Weyhrauch's program is its own theory of itself, so that it can reason about itself and as its model of its description of itself. Brown's program seems similar in basic nature. Smith [1978] is also working towards developing another such program and formal system. All these studies, however, are essentially foundational. None tell how to reason about oneself, but instead concentrate on providing the power to do so if one so desires. An aim of this thesis is to explain ways of doing just that, of using these frameworks for self-referential reasoning.

#### 1.7.2.2 The Nature of Reasoning

The mathematical semantics of the non-monotonic justifications and default inferences used in RMS and other programs has recently been developed by McDermott and Doyle [1978]. Reiter [1979] analyzes a less general system allowing stronger results while still capturing many important inferences. Kramosil [1975] is the first, but unfulfilled, study of this sort. McDermott [1980] follows up our earlier approach with stronger logics based on traditional the modal logics T, S4, and S5. I suspect there may be another interesting logic along these lines, namely a non-monotonic extension of Boolos' modal logic of provability in Peano arithmetic [Boolos 1979]. Reiter [1978] catalogs some of the many important appearances of non-monotonic reasoning in artificial intelligence studies.

Another close relative to non-monotonic logic and these views is the theory of conclusions as formulated by Tukey [1960] and devloped by Dacey [1978]. This is a formal logic in which statements with very strong evidence can be adopted as conclusions, to be maintained independently of the evidence until and unless very strong evidence to the contrary is accepted.

Harman, Lehrer and Paxton, Scriven, and Bennett each formulate view of inference which seem close in some ways to non-monotonic inference. Harman [1973] sees inferences as total views, with all inferences containing the proviso "and since there was no undermining evidence." Lehrer and Paxton [1969, Lehrer 1974] formulate knowledge as undefeated justified true belief. Scriven [1959, 1963] formulates historical explanations as involving truisms or what he calls "normic rules" which state "true" general principles which may be defeated in particular instances. He argues that such rules are neither deductive nor statistical in nature. Bennett [1964] develops a notion of "R-denials" as denials of reasons, but apparently does not continue the process with denials of denials in any uniform way.

Rosenberg [1978] presents a beautiful exposition of the conversational logic of dialectical argument. Belnap [1976] discusses a simple four-valued logic of this sort of argument, and shows its connection to relevance logic.

There is a sizable literature on logics of various attitudes, such as belief, desire, and obligation. Rescher [1968] surveys this area. See for example Hintikka [1962], Kenny [1978], Rescher [1966], [Hilpinen 1971] and Chisholm [1978]. For the most part, I have not developed a comprehensible relationship between these logics of attitudes and the behaviors of the program suggested here. The formal logics all seem too simplistic, or specialized to very specific sorts of reasoning. Chapters 4 and 5 briefly mention some connections of the ideas proposed here with deontic logic.

Kreisel [1968, 1971, 1977] and Prawitz [1973] survey the literature and ideas of proof theory, and Boolos [1979] presents the correct modal logic of provability in arithmetic. Proof theory is intimately related to reasoning about reasoning, it being in large part formal reasoning about formal reasoning systems. Closely related, intuitionists reflect on the structure and development of proofs as a means of judging what arguments are constructive or non-controversial. See [Heyting 1956] and [Yessinin-Volpin 1970].

Collins [1978] and Wason and Johnson-Laird [1972] discuss questions in human plausible reasoning and the psychology of reasoning. I am not yet familiar enough with this literature to comment on it.

Our approach to reasoning should be taken as orthogonal in many ways to the decision-theoretic approaches mentioned below, and to Zadeh's fuzzy logic, which aims at capturing a separate set of intuitions. See [Zadeh 1975] and [Gaines 1976].

#### 1.7.2.3 The Theory of Intentional Action

Shaffer [1968], Taylor [1966, 1974], and Davis [1979] survey the standard theories of intentional action. White [1968] and Brand [1970] collect a number of papers on this topic. See also [Anscombe 1957] and [Goldman 1970].

Miller, Galanter, and Pribram [1960] discuss the role of plans in psychological explanations of behavior, and Collingwood [1946] the role of intention in historical explanation. Dray [1964] surveys theories of historical explanations, and Gardiner [1974] and Hook [1963] collect a number of papers in this area.

Studies of program understanding and action interpretation develop a number of models and techniques for representing plans, recognizing plans in programs, devices, or patterns of behavior, and analyzing errors to find the faulty plans that caused them. See Sussman [1975], Goldstein [1975], Brown [1976], de Kleer [1979a], Miller [1979], Rich and Shrobe [1976], Wilensky [1978], and Schmidt, Sridharan, and Goodson [1978].

#### 1.7.2.4 The Fragmentation of Values

Nagel [1979b], Fodor [1975], and Putnam [1975] present arguments for the irreducibility of the various domains of the world to a common basis of comparison. The basic arguments are that even if we are able to determine how each of the domains is realized in a more fundamental domain, these reductions cannot be lawlike because there are so many sorts of ways of realizing each domain in the underlying domains, and that even if they were lawlike, the bridging realization explanations would be so hopelessly detailed that they would never make sense in arguments, reasoning, or decision-making. When each person is taken as a separate domain of values, as is usual in social decision-making, there result a number of problems due to the fragmentation of values. Arrow [1967] discusses the fundamental result of the nonexistence of a "nice" way of combining fragmented values coherently in all cases to find an aggregate value.

### 1.7.2.5 Decision-making

Much recent work in decision-making has been developed in decision-theory, the most popular branches of which are based on Bayesian probability theory, and most of this work concentrates on chauvinistic utility measures. Suppes [1967] surveys this area. Good [1952] mentions a hierarchical decision-theory of this sort. Duda, Hart, and Nilsson [1976] apply these ideas in the context of popular AI techniques. Giles [1976] develops a subjective logic of belief along these lines. Simon [1976] introduced the notion of satisficing to avoid hopelessly idealized rationality. Allison [1971] and Braybrooke and Lindblom [1963] discuss social and political models of decision-making.

Many studies have been made on various structures for organizations and decision-making in them. Many of the ideas and concerns here are closely connected with those of the control and organization of reasoning programs. See for example Barnard [1938] (which has an intriguing appendix on the nature of mind and reasoning, logical and non-logical), Drucker [1946, 1974], Simon [1976], March and Simon [1958], Chandler [1962], Rawls [1971], and Nozick [1974]. Related studies attempt to view animal and human behavior as stemming from organizations of smaller decision-making units. Tinbergen [1951] presents such control structures for several animals. Minsky and Papert [1978, Minsky 1977] explore such organizations for the human mind. Fox [1979] attempts to relate AI decision-making models, organization theory, and decision theory.

Quite different from that on decision theory, the literature on deliberation usually admits the fragmentation of values, and concentrates on the reasons involved in the deliberation. See the articles in Raz [1978], and books by Aune [1977], Castaneda [1975], Edgley [1969], Gauthier [1963], Hare [1952, 1963], Harman [1977], Nagel [1970], Norman [1971], and Richards [1971].

### 1.7.2.6 Control of Reasoning

Basic studies of controlling actions by constructing and executing plans of action include Newell and Simon [1963], Ernst and Newell [1969], Fikes and Nilsson [1971], Fahlman [1974], Sacerdoti [1974, 1977], and Tate [1977]. Sacerdoti [1979] surveys these techniques. As mentioned earlier, planning techniques have been applied to controlling reasoning as well by Hayes [1973b], Doyle [1976], de Kleer et al. [1977], and McDermott [1978]. Latombe [1976, 1979] and Stefik [1980] take this approach as well.

Gordon et al. [1978] develop a proof-construction system based on an explicit language/metalanguage distinction, and encourage the encoding of proof construction strategies as metalanguage programs. However, they leave all planning to the human user, and do not self-apply the program. In particular, their system never records proofs, and hence cannot reason about its own reasoning.

In the "pure" production system framework, McDermott and Forgy [1976] discuss techniques for conflict resolution and focus of attention. Rychener [1976] presents an interesting implementation of GPS in such a production system. Hayes-Roth and Lesser [1977] explore "focus of attention" techniques in a "blackboard" production-system architecture. In the "deductive" production system framework, Davis [1976, 1980] developed meta-rules as a way of encoding control information. In all these approaches, however, control depends on a chauvinistic decision-making technique that operates without reasons, and neither approach involves a particularly coherent notion of action.

A final approach (or non-approach) is that of the logic programming community. Kowalski [1974] seems content to refuse the problem of control as a domain for reasoning. Pratt [1977] seems to beg the question by concentrating on the "facts" and postulating an intelligent interpreter to decide what to do with them, much like the earlier GPS and mechanical theorem proving methodologies.

#### 1.7.2.7 Adaptive Changes of Mind

Russell [1930], Carnegie [1936, 1944], Ellis and Harper [1961], and Johnson [1977] discuss informal techniques for changing ones attitudes in the context of self-improvement. There is a large literature on this problem, but these are the best expositions I have seen. Suppes [1977] surveys several influential learning theories.

Concept learning is discussed by Winston [1975] and Fahlman [1979].

Belief revision has been an active field recently, and the literature is surveyed and indexed by Doyle and London [1980]. Hayes [1973a] is still an excellent earlier survey. My approach in [Doyle 1979] has close relatives in the works of London [1978], McAllester [1978], Thompson [1979], Fikes [1975], and Stallman and Sussman [1977]. de Kleer and Harris [1979] critically compare these approaches. Charniak, et al. [1979] present a simple RMS in explicit detail with considerable discussion. London applies this approach in detail to belief revisions following actions. Fahlman [1974] and Sridharan [1976] present schemes for describing rules to disambiguate action effects, their common suggestion being rules which choose one revision over another on the basis of aspects of the particular beliefs being revised. Some approach of this sort is necessary because revisions due to inconsistencies and actions can typically be done in many ways, so some way of choosing between the alternate revisions must be possible. Excellent general of belief revisions can be found in Rescher [1964, 1976], who presents a formulation of consistency-based belief revision, and in Quine [1953], and Quine and Ullian [1978], who discuss the ambiguity of revisions and several sorts of general guidelines for disambiguating them. Goodman [1973], Lewis [1973], Turner [1978], and Rescher [1976] study counterfactual and plausible reasoning. Analyses of counterfactuals usually involve some way of evaluating the consequent of the counterfactual statement in circumstances as "close" as possible to the actual circumstances but in which the hypothesis of the counterfactual holds. These proposals for counterfactuals thus suggest ways of choosing "minimal" revisions of beliefs to accommodate new hypotheses. Sosa [1975] collects a number of papers on this topic.

Fahlman [1974], Sacerdoti [1977], and Shrobe [1979b] discuss revision of one's plans.

Harper [1976] discusses changes of preference in a probabilistic setting.

Sussman [1975] studies the problem of skill development. Fikes and Nilsson [1972] discuss the collection of STRIPS plans, and Davis [1976] the acquisition of new inference rules.

#### 1.7.2.8 Affect and Intellect

Freud [1937] analyzed the impact of affect on intellect through repression and censors. Ellis and Harper [1961] base their psychotherapy on the converse influence of intellect on affect. They analyze people's problems by finding the troublesome statements the afflicted repeat to themselves. Minsky [1980] explores how affect and intellectual activities are aspects of the same mechanisms, how affect exploits intellect for its purposes, and how intellect similarly exploits affect.

#### 1.7.2.9 Consciousness

The standard positions on the nature of consciousness are surveyed by Shaffer [1968], Taylor [1974], and Dennett [1978a]. Other topics in the philosophy of mind and psychology are discussed in [Fodor 1968, 1975], [Gustafson 1964], [Glover 1976], [Dreyfus 1979], [Nagel 1979c, 1979d], [Boden 1977], Ryle [1949] and Dennett [1969, 1978c].

#### 1.7.2.10 The Absurd

Nagel [1979a], Quine [1953], Camus [1955], Sartre [1956], Anderson [1975], Wheeler [1977], and others discuss the problems of why we are the way we are, and why we should adapt. Pascal [1971], James [1971], and Kierkegaard [1944] discuss leaps of faith.

# CHAPTER 2

# THE REPRESENTATION OF STRUCTURE

One important kind of human action is that of building new things out of previous things. There may be little to distinguish a new thing from its components or its surroundings but our calling it so (as detractors of modern sculpture have been wont to point out). Nevertheless, we often find it useful to think of portions of the world as things constructed from other things. This chapter outlines a representational system designed to allow a program to share this way of thought.

Now conceivably, a program could build and use things and never think of them except as their constituents. This, however, has the disadvantage of unnecessary detail. It is ridiculous to think of moving a table across the room only in terms of the motions of individual molecules making up the table, or of a mind or machine only in terms of the physical events associated with its physical realization, but that would be a consequence of an inability to think of structures as objects, abstracting away all the unwanted details of their structure. Instead, the program must be able to think of its creations in terms other than their constituents. Since the program thinks about its internal actions as well as its external actions, we conclude that it should be able to make new representations out of previous representations, and then be able to use the new representations as objects in creating further representations.[23]

Often in physical constructions, the constituent parts retain their structure so that the structure of the whole includes the former structure of the parts. Of course, this is often not so, as in chemical mixtures or plastic deformations of constituents, for example salt dissolved in water and ice floes made into an igloo. But retained structure, when it exists, makes descriptions of constructs much simpler to comprehend, so we require further of the representational system that it allow structure retention when

[23] Harrison [1978] emphasizes the unity of the building activities involved in creative thought with the building activities involved in mundane constructions and practical reasoning. Lenat [1977] makes a similar point and presents a program for inventing mathematical concepts.

possible. In cases in which (internal and external) building operations leave intact the combined (internal and external) objects, this means that the structure of the representation reflects the structure of its referent.

We also place some distinctly non-physical requirements on the representational system.

The previous chapter made many arguments in support of the program's ability to explain its structure and behavior, and the representational system should make this possible. We require that each representation include information explaining how it was formed from other representations, and what processes were responsible for its formation.[24]

Another important requirement is the ability to economize on the storage size of representations. To consider an analogous case, large corporations must often raise large sums of money, much larger than they might borrow directly. They do this by borrowing from a number of banks, who in turn borrow from other sources. Jack borrows from Jill and Jane, who borrow from John and Jake and James and Jonas, who borrow from Jean and Joan and others, so that many of the effective funds are only virtual possessions, not a single actual bank-account. In a similar way, the program should economize on the information for which it actually uses long-term storage resources. It can do this by using the records of how representations were constructed from others to temporarily reconstruct the apparent structure of a representation when answering questions, and then to discard all but the basic information about the representation and its structure.

[24] It would be nice if representations explained not only the how but the why of their formation. Unfortunately, as the last chapter speculates, it may not be possible always to say why. This question depends on the completeness of the program's self-description, on its knowledge of itself being detailed enough to tell the purpose of each of the actions of its procedures.

## 2.1 Desiderata of the Representational System

In summary, the desiderata for the representational system, along with examples and how we realize them, are as follows.

*1. The representational system should be able to represent all the objects considered by the program.*

This requirement has two parts. The first is the simple semantical adequacy of the representational system, which rules out, for example, a representational system whose only symbol is the numeral 3, for 3 is just one symbol, and there are many things which must be represented simultaneously. We adopt a system based on the first-order predicate calculus (FOPC), as it is the best understood formal representational language. However, this choice is intended to be the most colorless choice possible. Since no one has yet actually demonstrated the adequacy of any known representational system (FOPC included) for describing everything, we take FOPC as a base for extension, such as modalities, etc., and do not address completions or alternatives of this language.

The second part of this requirement concerns the physical realization of the representational system. A purely formal system cannot represent anything, for what it thinks of as its representing something is not supported by actual causal connections between its thoughts and its objects. Several authors, such as Putnam [1978], Fodor [1978], and Searle [1980], discuss this issue in detail. We do not discuss this question further, and take for granted a realization of the representational system as part of a machine actually connected to the physical world in the proper ways.

*2. New representations can be built from previous representations.*

The basic unit of representation in a FOPC-based system is the logical theory, or set of statements. This requirement means that we can combine sets of statements to get new sets of statements. We do not restrict these combinations to be simple unions of the sets, but can make more complicated, non-additive combinations. But a simple mathematical example is that of combining a theory describing a set of objects as a group under one operation and another theory describing a subset of those objects as a group under another operation into a theory describing the objects as a semiring.

*3. Combinations of representations are objects as well.*

This means that the representational system treats sets of statements as objects to which statements can refer. For example, one might have a theory describing a semiring theory as a combination of two other theories. Here the first theory treats the other three as objects.

*4. Each representation incorporates an explanation of how it was constructed.*

This means that the theories and their statements include the reasons mentioning the other theories, statements, and procedures which constructed them. Precise explanations of this will largely be deferred until Chapter 3. But an example might be a theory constructed by adding together two other theories. The statements in this theory would all have reasons mentioning the corresponding statements in the initial theories, along with the statements relating the combination theory and the constituent theories, and finally, along with the procedure which inferred the new statements from the earlier statements and the theory-construction statements.

*5. The representation is asymptotically storage-space efficient.*

This requirement means, for example, that statements in a theory are not actually inferred from the constituent theories unless actually needed, and are not retained unless needed.[25]

In the remainder of the chapter, we will base the representational system on *virtual copies*

[25] In [Doyle 1977] I suggested that asymptotic storage-space efficiency was a major factor in the design of representation languages intended for use in representing human-sized bodies of information about the world. I also argued that virtual-copy representational systems like Fahlman's NETL [Fahlman 1979] are best viewed as attempts at asymptotic storage-space efficiency.

(VC's), a term due to Fahlman [1979]. Virtual copies of theories will be theories whose statements can be inferred, when needed, and discarded when not required. Virtual copies can be modified by adding in other, non-virtual statements, and by defeating some of the virtual statements. This last capability is used for describing overridden defaults, exceptions, and what might be called family resemblances, in which the simplest way of describing a number of objects is as a number of distinct modifications of an ideal family member. As we will describe in more detail later, VC inferences are non-monotonic inferences, to allow these sorts of non-additive theory modifications.

Unfortunately, the claim that the program uses virtual copies is a fiction. All the versions of it that I have implemented in fact make actual copies, that is, always permanently infer all statements of all theories. However, this is merely an accident of time pressures on my implementation efforts, as the full-copy techniques are easy to implement quickly, and the virtual-copy techniques are harder to implement correctly, as there are many subtleties involved.

This fiction about the representational system presented here is actually a symptom of a larger incompleteness in this thesis, namely the lack of database retrieval procedures altogether. McDermott, Fahlman, and others have argued for a separation between database retrieval and problem solving, where database retrieval consists of applying automatic, quick procedures which adequately handle almost all queries (the routine cases), and problem solving consists of applying carefully controlled inference procedures to ferret out the desired information that the routine procedures miss. This distinction is sometimes hazy, but is a convenient way of viewing the problem, and I adopt it here. Routine retrievals are carried out by a set of standard, efficient, but sometimes inadequate database interrogation procedures. The difficult cases are handled by self-applying the reasoner with means of information retrieval plans and deliberation about where to look for information. This thesis discusses neither the routine procedures nor the information retrieval plans. The representational system presented here is capable of reinterpretation as other representational systems, for example, as NETL, and retrieval algorithms developed for them can easily be adapted to the data-structures used here. Likewise, the ability of the program to refer to its own representations allows formulation of information-retrieval plans for careful reasoning.

One final introductory remark: This chapter is not intended as a presentation of the classical open problems of representational theory. The system presented here can be viewed as a simple extension of the ideas of Fahlman and Weyhrauch [1978] to include reasons for representations. Smith [1978] describes how many classical representational puzzles can be fruitfully attacked with representations which can be referred to as objects by other representations. Both Hayes [1977b] and Nilsson [1980] present alternative readings of hierarchical representational systems as non-meta-theoretical FOPC systems, but their readings have major semantical shortcomings, discussed in Section 2.5.

## 2.2 A Key Application

The program uses a library of hierarchically organized plans and primitives. It occasionally builds new plans and adds them to this library. For example, it might make a plan for cooking a single spaghetti dinner from two existing plans, that of cooking and refrigerating a vat of spaghetti sauce, and that of heating some spaghetti sauce and cooking some spaghetti. To construct the new plan, it concatenates the two existing plans, changing the quantities involved, and removing the steps of refrigerating and reheating the sauce. To do this, it makes copies of the representations of the previous plans, identifies some of the components of these copies, deletes some of their components, and then packages up the resulting collection as the new plan.

We view these steps of copying and modifying representations in terms of the above requirements as follows. The program first creates the copies of preexisting plans by making new representations along with inference rules which make the assumption that any part of a prototype representation is also part of the corresponding copy representation. These inferences are non-monotonic assumptions, so that modifications may be made to the representation by defeating the assumptions. The identifications are accomplished by creating inference rules which duplicate any conclusion about one representation with similar conclusions about the identified representations. Finally, the collection of modified and interconnected representations is reified as a new plan representation available for further copying and combination. The rest of this chapter presents the details of these operations.

### 2.3 SDL, a Structured Description Language

The program employs a representational language called SDL. SDL is based on a predicate calculus, but bears strong resemblances to current structured-description representational systems. In particular, SDL involves both a modified form of the data-structures of FOL [Weyhrauch 1978] and a particular way of using these data-structures based on NETL.

The basis of SDL is the first order predicate calculus. However, where normal FOPC systems are viewed as having one language, one set of axioms, and a model external to the language, SDL employs many languages, axiom sets, and models simultaneously. It describes each object with a separate set of axioms in an appropriate language and its intended model. SDL describes the structural relationships between such descriptions by treating each of these logical theories as an individual with parts. These meta-theoretic relationships then become axioms of yet other logical theories.

The most important data-structure in SDL is the *theory*. The standard usage of "theory" in mathematical logic is the set of theorems of some set of axioms in a formal language, that is, the axioms together with all their logical consequences. Following Weyhrauch, we corrupt the usage of this term to mean a data-structure combination of a language definition, a set of facts (axioms and theorems) in the language, and a simulation structure (partial model) for the set of facts and the language, or mnemonically, $T = \langle L,S,F \rangle$. We explain all of these components below.

All of SDL's first order languages are constructed from the standard logical connectives along with individual constants and variables, predicates, functions, and predicate and function parameters (for axiom schema). In addition, the languages are many-sorted, with a system of partially ordered sorts. (In logic, the term "sort" means kind-classification, not ordering classification.) In many respects the system of sorts is an inessential convenience of the languages, although they turn out to be nontrivial extensions computationally. Other kinds of extensions to the type of language allowed, such as modalities and conditional expressions, are not used or explored here for simplicity, and might be added in future versions of the program.

We define a language in SDL by specifying the non-logical symbols in the language and the roles of these symbols. Language definitions consist of the following types of declarations. The first argument of these commands, name, is always a Lisp atomic symbol or a pathname (explained later). Types are also Lisp atomic symbols, which are defined as predicate constants of the language. Theory's are the theory data-structures in whose language name is being defied. The number of arguments are, when specified, non-negative integers (Lisp integers). Argument names and types are Lisp atomic symbols defined as individual variables and predicate constants of the language. Likewise, result types are sort predicate constants of the language. The last argument is a justification (as explained in the next chapter) used as the reason for the data-structures created by the declaration.

```
(INDIVIDUAL-CONSTANT name type theory justification)
(INDIVIDUAL-VARIABLE name type theory justification)
(PREDICATE-CONSTANT name {# of args} {({arg name} arg type) list} theory justification)
(PREDICATE-PARAMETER name {# of args} {({arg name} arg type) list} theory justification)
(FUNCTION-CONSTANT name {# of args} {({arg name} arg type) list} {result type} theory justification)
(FUNCTION-PARAMETER name {# of args} {({arg name} arg type) list} {result type} theory justification)
```

In the following we write these commands in a syntax similar to FOL's. In this syntax, the theory is given by the context of the presentation. The statement "IN theory" is used to switch attention to the theory with the global name theory. (Once we have defined them later on, we will allow pathnames as well.) We usually ignore justifications for simplicity of exposition.

For example, we might construct a language for discussing natural numbers and arithmetic with the declarations:

```
IN ARITHMETIC:
        Function-constant SUCCESSOR 1 NATNUM;
        Function-constant PREDECESSOR 1 NATNUM;
        Function-constant + 2 (NATNUM NATNUM) NATNUM;
        Function-constant * 2 (NATNUM NATNUM) NATNUM;
        Predicate-constant < 2 (NATNUM NATNUM);
```

These declarations define the usual symbols of successor, predecessor, plus, and times, and the ordering predicate.

We use SDL to discuss not only languages, but their models and their relations to their models as well. However, many intended models involve objects which simply do not exist inside a computer, for example, cows, real numbers, and redness. Because we can sometimes present the elements of models inside the computer and sometimes not, instead of ordinary models we employ *simulation structures.* A simulation structure can be thought of as a partial model, one which includes partial decision procedures to represent its domain and the set of constants, and a set of attachments. We take these decision procedures to be Lisp procedures which take an object as input and tell whether or not it is one of the objects in the domain (constant) or domain (constant) representation. The list of attachments is essentially an association list pairing linguistic symbols with domain elements as their referents, thus specifying the set of "bindings" of the symbols to objects in the model. A simulation structure may not completely determine the truth value of every statement in the language, but it may determine the truth value of some. This is as good as we can hope for, and is all we will require. Attachments are made with the command

```
(ATTACH name object theory justification),
```

domains and constants with the command

```
(REPRESENT name representation theory justification).
```

The ATTACH command adds the specified pairing to the list of attachments of the simulation structure, with the given justification. The REPRESENT instruction declares the name to be a predicate and sort symbol of the language and attaches the name to the representation function in the simulation structure.[26] One particular sort of attachment is that of a procedure in a theory, in which an individual constant is attached to a LISP procedure. All procedures are named by such attachments, so that values computed by them may be justified in terms of the procedure as the "inference rule".

Of course, one does not have a model of a language, but rather a model of a set of statements in the language. These statements are called facts (to subsume both axioms and theorems), and are declared by either

```
(AXIOM name wff theory justification)
```

or

```
(FACT name wff theory justification).
```

Each of these facts is added to the set of facts of the theory. Each fact consists of both the name of the fact (a symbol in the theory's language), and a wff of the language of the theory. This connection between fact name and wff is treated as an attachment of the theory, although here the attachment is from a symbol of the language to a wff in the set of axioms and theorems. Thus theories with axioms refer to parts of themselves.

Theories are made up out of a language, a simulation structure, and a set of facts. Theories are created with the command

[26] This chapter will be hazy on exactly what representations are and how they relate to languages and simulation structures. The intended ideas can be illustrated with numbers. One has the numerals in the language, which refer to numbers, and since numbers don't exist in the computer, we add in Lisp fixnums as a representation of numbers. The distinction becomes important because in many cases, the program will have the referent for a symbol, namely a data-structure which does exist inside the computer. (Actually, the existence of data-structures in the computer may be a fiction. Data-structures are referred to by pointing to some location in memory, but the intended data-structure results only through interpretation of the information in that location as further pointers, fields, etc. In this way, the fiction of data-structures is much like the fiction of the "self" of the program, since the program is one big data-structure interpreting itself.) Weyhrauch and others discuss the problem of languages, models and representations, and I expect to adopt one of their suggestions when I become more familiar with their proposals.

```
(THEORY name parent-theory justification),
```

which declares name to be an individual constant in the parent theory, creates a new theory data-structure, and attaches this data-structure to name in parent-theory.

For example, we can declare more of the theory of natural number arithmetic as follows.

```
IN ARITHMETIC:
          Individual-constant 0 natnum;
          Individual-variable n natnum;
          Individual-variable m natnum;
          Predicate-parameter P (natnum);
          Axiom Oneone: ∀n ∀m successor(n)=successor(m) ⊃ n=m;
          Axiom Succ1: ∀n ¬0=successor(n);
          Axiom Succ2: ∀n [¬0=n ⊃ ∃m n=successor(m)];
          Axiom Plus: ∀n [n+0=n ∧ ∀m[n+successor(m)=successor(n+m)]];
          Axiom Times: ∀n [n*0=0 ∧ ∀m[n*successor(m)=(n*m)+m]];
          Axiom Induct: [P(0) ∧ ∀n[P(n) ⊃ P(successor(n))]] ⊃ ∀nP(n);
          Attach Successor (LAMBDA (X) (ADD1 X));
          Attach Predecessor (LAMBDA (X) (COND ((> X 0) (SUB1 X)) (T 0)));
          Attach + +;
          Attach * *;
          Attach < <;
          Attach 0 0;
```

The first two attachments above attach Lisp procedures to two predicate constants of the theory.[27] The next four attachments attach to a symbol of the theory the value attached to the same symbol in the global theory. In the first three of these, the value is a Lisp procedure, and in the last it is the Lisp number 0.

Each of these data-structures contains information about the reasons for the data-structure, which are stored as justifications for a RMS node, as explained in Chapter 3. Each theory data-structure has a justification mentioning the procedures which created it. Each declaration of a linguistic symbol adds a justification to that declaration. Each attachment has a justification, and so does each axiom in the theory. That is, an axiom would have a premise justification in the theory, but that premise justification itself would not be an assumption, but would have a justification specifying the reason for this fragment

[27] Those familiar with SCHEME [Steele and Sussman 1978a] should understand that we ideally would employ SCHEME instead of LISP, so that these attached values would be procedures (closures) rather than s-expressions.

of the theory in terms other theories and inference procedures. As usual, consequences have justifications mentioning both the nodes of their antecedents and the inference rule or procedure deriving the consequence.

We will represent all of these things with the following data-structures. We notate these in the "structure" syntax of MIT Lisp Machine Lisp [Weinreb and Moon 1979], in which a name is specified followed by the fields of the data-structure. The first structure defines the fields common to all the rest: the name, the RMS node, and the parent (whose function is explained following these definitions). The :INCLUDE specification is the means by which these common field definitions are included in all other structures.

```
(DEFSTRUCTURE (COMMON-STRUCTURE)
 NAME
 NODE
 PARENT)
```

These declarations define the data-structures associated with languages.

```
(DEFSTRUCTURE (LANGUAGE (:INCLUDE COMMON-STRUCTURE))
 INDIVIDUAL-VARIABLES
 INDIVIDUAL-CONSTANTS
 PREDICATE-CONSTANTS
 PREDICATE-PARAMETERS
 FUNCTION-CONSTANTS
 FUNCTION-PARAMETERS)

(DEFSTRUCTURE (INDIVIDUAL-CONSTANT (:INCLUDE COMMON-STRUCTURE))
 INDIVIDUAL-TYPE)

(DEFSTRUCTURE (INDIVIDUAL-VARIABLE (:INCLUDE INDIVIDUAL-CONSTANT)))

(DEFSTRUCTURE (PREDICATE-CONSTANT (:INCLUDE COMMON-STRUCTURE))
 NUMBER-OF-ARGUMENTS
 ARGUMENT-TYPE-LIST)

(DEFSTRUCTURE (PREDICATE-PARAMETER (:INCLUDE PREDICATE-CONSTANT)))

(DEFSTRUCTURE (FUNCTION-CONSTANT (:INCLUDE PREDICATE-CONSTANT))
 RESULT-TYPE)
```

```
(DEFSTRUCTURE (FUNCTION-PARAMETER (:INCLUDE FUNCTION-CONSTANT)))
```

These declarations define the data-structures associated with simulation structures.

```
(DEFSTRUCTURE (SIMULATION-STRUCTURE (:INCLUDE COMMON-STRUCTURE))
 DOMAIN-REPRESENTATION
 CONSTANTS-REPRESENTATION
 ATTACHMENTS)

(DEFSTRUCTURE (REPRESENTATION (:INCLUDE COMMON-STRUCTURE))
 REPRESENTATION-ALGORITHM)

(DEFSTRUCTURE (ATTACHMENT (:INCLUDE COMMON-STRUCTURE))
 OBJECT)
```

This data-structure is used for facts.

```
(DEFSTRUCTURE (FACT (:INCLUDE COMMON-STRUCTURE))
 WFF)
```

This structure defines the data-structure for theories.

```
(DEFSTRUCTURE (THEORY (:INCLUDE COMMON-STRUCTURE))
 (THEORY-LANGUAGE (MAKE-LANGUAGE))
 (THEORY-SS (MAKE-SIMULATION-STRUCTURE))
 THEORY-FACTS
 EQ-POLICIES
 EQ-POLICY-LIST
 VC-TYPE-THEORY
 VC-UP-STATEMENTS
 VC-DOWN-STATEMENTS
 VC-STATEMENTS-LIST
 MAP-UP-STATEMENT
 MAP-DOWN-STATEMENTS
 MAP-STATEMENTS-LIST)
```

In the above, the structures (particularly Theory) contain not only the lists we have previously indicated, but also slots for redundant forms of these lists to facilitate retrieval and manipulation of information. The basic such slot is that of PARENT, which typically is used as a reverse pointer from a sub-data-structure to the data-structure which includes it. The exact interpretation of this slot varies with the data-structure involved. Languages, simulation structures, facts, and reasons point back to their theory; individual constants, individual variables, predicate constants, predicate parameters, function constants, and function parameters point back to their language; representations and attachments point back to their simulation structure, and theories point back to the theory which is their context of existence.

THEORY, in addition, contains slots to facilitate retrieval of structure-sharing statements, part inferences, and dataflow policies. These will be explained later in this chapter and in Chapter 4.

This completes the description of the underlying logical system.

## 2.4 How to use SDL

We represent objects hierarchically in SDL by using a separate theory to describe each object. The parts of the object are in turn described by other theories, and the theory of the object includes statements of the relations between these parts and between their theories. When two objects are mutually defined, each of the theories describing these objects will contain the other theory as a part. This means of representation is not paradoxical because the theories of the parts are copies of their prototype theories.

For example, suppose we wish to describe as objects arithmetic relations between numbers. To do this, we can make a theory ADDER as follows.

```
IN ADDER:
        Individual-constant A1;
        Individual-constant A2;
        Individual-constant SUM;
        Predicate-constant =;
        Function-constant +;
        Attach + +;
        Attach = =;
        Axiom Plus: A1+A2=SUM;
```

This theory describes the prototypical adder. ADDER has three individual constants for the addend, augend, and sum, and, via attachments to the arithmetic predicates and functions, defines the relation between the constants.

Notice that the description of the prototype contains no attachments to the constants. That is because the prototypical adder does not relate any particular numbers or have any default values. Suppose we wish to make an instance of this description for the addend and augend values 3 and 4. We first would create a new theory which is a virtual copy of ADDER, namely,

```
IN T-1:
        Individual-constant T-1;
        Attach T-1 T-1;
        Individual-constant ADDER;
        Attach ADDER ADDER;
        Axiom VC(T-1, ADDER);
```

T-1 is the theory's name in the global theory. T-1 is also the theory's name for itself. ADDER is the theory's name for the theory with the global name ADDER. The sole axiom in T-1 allows us to make a number of conclusions within T-1. The VC inference rule is that all statements defining a theory, including the language, the simulation structure, and the facts, are inferred in the copy theory as individual non-monotonic assumptions. That these inferences are non-monotonic will be important later when we wish to modify the copies of prototypes to override default values or to describe exceptions. Thus T-1 actually has the following statements.

```
IN T-1:
        Individual-constant T-1;
        Attach T-1 T-1;
        Individual-constant ADDER;
        Attach ADDER ADDER;
        Axiom VC(T-1, ADDER)
        Individual-constant A1;
        Individual-constant A2;
        Individual-constant SUM;
        Predicate-constant =;
        Function-constant +;
        Attach + +;
        Attach = =;
        Axiom Plus: A1+A2=SUM;
        Attach A1 3;
        Attach A2 4;
```

To this, we have added the two values as attachments to A1 and A2. By use of the axiom PLUS of this theory, the attachments can be used to compute an attachment for SUM to the value 7.

The idea of VC theories could also have allowed writing ADDER more succinctly, by declaring ADDER to be a VC of ARITHMETIC. In this way, ADDER would have been an extension of ARITHMETIC, and the extra definitions of +, =, etc. would have been unnecessary.[28]

However convenient might be theory extensions made in this way, many circumstances require a theory to contain as subtheories multiple distinct copies of other theories. The main motivation for this is the need to describe structures having several parts, each of the same type, but each having its own peculiarities. We facilitate this by means of the TYPED-PART command, as the next example shows.

We can make a new description, called DOUBLER, by modifying a copy of ADDER.

```
IN DOUBLER:
        Individual-constant X;
        Individual-constant 2X;
        Typed-Part ADDER ADDER;
        Axiom: X = [A1 ADDER];                      ;[Pathname] explained below.
        Axiom: 2X = [SUM ADDER];
        Axiom: [A1 ADDER] = [A2 ADDER];
```

The expressions in brackets are called *pathnames*, and are compound names treated as the corresponding names in the subtheories. That is, [A B ... C] should be interpreted as the variable A of the theory named B ... of the theory named C. We write V[pathname] to notate the value attached to the symbol represented by the pathname, so V[A B] is the value attached to A in the theory attached to B in the current theory.

The command

```
                    (TYPED-PART name prototype justification)
```

expands into several other statements and actions. It creates a new theory as a virtual copy of the

[28] The Edinburgh LCF proof construction system makes similar use of a collection of theories (sets of theorems) with its "ancestry graph." [Gordon et al. 1978]

prototype, and then creates a constant of the given name in the theory and attaches the copy to the name in the theory. Thus we have the new statements

```
IN DOUBLER:
        Individual-constant ADDER;
        Attach ADDER T-2;
```

where we have also created the theory T-2:

```
IN T-2:
        Individual-constant T-2;
        Attach T-2 T-2;
        Individual-constant ADDER;
        Attach ADDER ADDER;
        Axiom: VC(T-2, ADDER);
```

Now by itself, this new theory T-2 is not much good, since the original doubler theory can only refer to it, not use it. However, the final function of the TYPED-PART statement is to enable the inference rule that any statement of T-2 is also a statement of DOUBLER under a rewriting of names of T-2 into reference expressions in DOUBLER. With this rule, DOUBLER gets the new statements

```
IN DOUBLER:
        Individual-constant [A1 ADDER];
        Individual-constant [A2 ADDER];
        Individual-constant [SUM ADDER];
        Predicate-constant [= ADDER];
        Function-constant [+ ADDER];
        Attach [= ADDER] =;
        Attach [+ ADDER] +;
        Axiom [PLUS ADDER]: [A1 ADDER] [+ ADDER] [A2 ADDER] [= ADDER] [SUM ADDER];
```

Note here that all symbols in the language of the part-theory are replaced by pathnames when they are inferred in the whole-theory. However, the second items in attachments are not affected by these substitutions. Instead, those expressions are referentially opaque, as they are symbols in the language of the global theory, rather than symbols of the language of the part-theory.

Suppose we now wish to combine two doublers to get a quadrupler. This, of course, is straightforward.

```
IN QUADRUPLER:
        Individual-constant X;
        Individual-constant 4X;
        Typed-part D1 DOUBLER;
        Typed-part D2 DOUBLER;
        Axiom: X = [X D1];
        Axiom: [2X D1] = [X D2];
        Axiom: 4X = [2X D2];
```

Suppose, however, that we didn't quite want a quadrupler, but instead wanted to first quadruple and then add one. We could, of course, make something new using an extra adder with an attachment to 1 of one of its "inputs." But to show off the sort of local modification/exception idea, we instead make a local modification to the axiom of one of the doublers in the quadrupler.

```
T-3:
        Typed-part QUADRUPLER QUADRUPLER;
        Individual-constant 1;
        Attach 1 1;
        Cancel [PLUS ADDER D2 QUADRUPLER];
        Axiom PLS:  [A1 ADDER D2 QUADRUPLER] + [A2 ADDER D2 QUADRUPLER] + 1
                    = [SUM ADDER D2 QUADRUPLER]
```

The effect of the Cancel statement is to defeat the non-monotonic assumption of the specified statement. We then just add in the desired modification, and we are done. Alternatively, we could have switched the theory we were working with to the theory attached to ADDER in D2 in QUADRUPLER. We could have then just made the commands

```
IN V[ADDER D2 QUADRUPLER T-3]:
        Cancel Plus;
        Individual-constant 1;
        Attach 1 1;
        Axiom Pls:  A1 + A2 + 1 = SUM;
```

This shows how the statements inherited in one theory can be canceled.[29] We can easily represent default information in this way by using the non-monotonic nature of VC inferences. In fact, all of the

[29] Of course, there are limitations to this technique. An interesting example is that of a wagon being drawn by four horses, one of whom had one blind eye. This we might have said with ∃Horse in HORSES(Wagon) and ∃Eye in EYES(Horse) such that BLIND(Eye). Then RMS would have a pretty time finding a model, as it would have to pick one out of so many possibilities.

statements of copies of theories are assumptions, and can be defeated for reason. The distinction between what one considers to be default information and what one considers essential aspects of theories is entirely a matter of how willing one is to give up one statement rather than another. The program employs policies which guide decisions between alternate revisions of the its beliefs, as discussed in chapters 3 and 6. However, policies form merely the mechanism, not the vocabulary, of guidelines for revision of beliefs. Several authors, for example Fahlman, have proposed a trinary classification of the strength of attachment to beliefs in concepts, namely default, normal, and criterial (or essential). How these absolute classifications should be realized in policies is unexplored, although the obviously intended policies should at least say that any default statement should be rejected in favor of any normal statement, and any normal statement should bow before any criterial statement. I am not convinced that absolute, context-free policies of this sort are particularly useful, and so have not pursued them. I would much rather believe that each domain of reasoning has its own set of revision policies along these lines.

The above examples all used Typed-part to include theories defining objects in a theory. Another major application is that of including subtheories to define the sort predicates of the language. Unfortunately, I have not yet convinced myself of just how this should be done, whether by Typed-part, an analogue of it, or by direct VC inclusion. Part of my hesitation in this matter relates to yet another question unanswered here, that of how sort predicates are taken as defined in the first place. For example, the previous theory ARITHMETIC is often thought of as the definition of what natural numbers are, but the sort predicate enters that theory only as a relativizer on the variables. That is, the whole theory is of the form $\mathrm{NATNUM}(x) \supset \mathrm{AXIOMS}(x)$, and nowhere is there any statement of the form $\mathrm{AXIOMS}(x) \supset \mathrm{NATNUM}(x)$. I don't think this is a difficult problem to solve, but it is one that requires more attention than I have been able to devote to it.

## 2.5 Relations with other Representational Systems

Bless thee, Bottom! bless thee! thou art translated.
William Shakespeare, *A Midsummer Night's Dream*

It seems likely that SDL can be used to realize many of the current representation languages, although we do not demonstrate this here. For example, we can translate CONLAN [Steele and Sussman 1978c] into SDL. In this translation,

```
(CONSTRAINT name parts+types equivalences)
```

goes into a named theory, the parts of the theory being given by the parts and types, and the equivalences by equations. What we do not capture without further inference rules is the constraint language control and inference structure, which strives to propagate values through all the known relations between variables. On the other hand, we can add new parts to a theory at any time, which CONLAN cannot. Also, we can make theories like the following, which are far beyond CONLAN's expressive powers, since it does not subsume FOPC.

```
IN SANDWICH:
        Individual-constant BLOCK1 BLOCK;
        Individual-constant BLOCK2 BLOCK;
        Individual-variable MIDBLOCK BLOCK;
        Predicate-constant ON;
        Axiom: ∀ MIDBLOCK [ON(MIDBLOCK, BLOCK1) ≡ ON(BLOCK2, MIDBLOCK)];
        Axiom: ∃ MIDBLOCK ON(MIDBLOCK, BLOCK1);
        Axiom: ∃ MIDBLOCK ON(BLOCK2, MIDBLOCK);
```

This blocks-world theory describes the situation in which two blocks sandwich in a number of other blocks.

This theory, incidentally, also shows the distinction between individual variables and constants in a prototype. Constants refer to parts of the prototype, which are constant aspects of the prototype even if they seem like variable aspects in instantiations of the prototype. Variable are used only in general statements about the domain of parts of the prototype.

Hayes [1977b] and Nilsson [1980] present translations of representational systems like KRL [Bobrow and Winograd 1977] in FOPC, but these translations miss the point of most current representational systems. Hayes and Nilsson succumb to the temptation to confuse the ideas of description specialization and predicate subsumption.

Consider, for example, theories describing mammals and horses. We normally accept the statement $\forall x[\mathrm{HORSE}(x) \supset \mathrm{MAMMAL}(x)]$. We also might be likely to construct the theory describing horses (which contains the predicate HORSE) by refining with additional axioms the theory describing mammals (which contains the predicate MAMMAL). These are two separate connections between the predicates HORSE and MAMMAL, but Hayes and Nilsson confuse them. The reason they make this conflation is simply that without treating theories as objects, the only way they can approximate theory construction is with an implication.

This confusion has many severe problems. The first is the *family resemblance problem.* Consider a human family with several members. We might try to capture their commonalities of appearance by describing the prototypical member of the family. However, there may be no property (other than prototypical human properties) shared by all members of the family. Each member may have most of the properties described by the prototype, but be lacking a single property that all the other members possess. Now if we use SDL with its non-monotonic VC inferences, this circumstance presents no problem, and can be treated succinctly. But if theories are not objects, and the only tool available is implication, then the best that can be stated is that the prototype has the property

$$(P2 \wedge ... \wedge Pn) \vee (P1 \wedge P3 \wedge ... \wedge Pn) \vee ... \vee (P1 \wedge ... Pn\text{-}1),$$

which is hardly succinct. Hayes and Nilsson each allow default statements in the descriptions, which are essentially non-monotonic assumptions. But they cannot get the succinctness and freedom of description construction that SDL allows unless *each* statement is taken explicitly as an assumption, including all instances of the implications relating concepts.

## 2.6 Advanced Applications

I have not explored the full powers (or even the complete details) of this representational system, particularly the hard questions concerning modality, non-denotation, and existence. For example, suppose one had the theories

```
IN UNICYCLE:
        Typed-part WHEEL WHEEL;                   ;etc.

IN WHEEL:
        Typed-part TIRE TIRE;
        Typed-part HUB HUB;                       ;etc.
```

and wished to say that some WHEEL-1 had no tire. If no attachment is made to [TIRE WHEEL-1], that would just be a lack of information about the question, not a definite belief that WHEEL-1 had no tire. However, one could state

$$\neg\exists x \text{ ATTACHED([TIRE WHEEL-1], x, WHEEL-1),}$$

which would seem to say that the term [TIRE WHEEL-1] lacked a referent. I have not yet been able to explore in detail whether this sort of trick can be used to attack the classical problems of existence and proper names, as in "Pegasus does not exist." Would the domains of existence be specified by the theory in which the nonexistence statements occurred? For example,

$$\neg\exists x \text{ ATTACHED(Pegasus, x, REAL-WORLD-THEORY),}$$

but

$$\exists x \text{ ATTACHED(Pegasus, x, MYTHOLOGY-WORLD-THEORY).}$$

Consult [Smith 1978] and [Martin 1979] for more detailed treatments of these sorts of puzzles in hierarchical structured representational systems, and [Haack 1978] and [Linsky 1977] for surveys of the classical problems.

## 2.7 Theories about Theories

As the preceding examples suggest, theories may be constructed to describe not only objects in the external world, but equally important, other theories. Thus the preceding theories typically described not only their "proper" subjects, but also their relations to other theories. For example, DOUBLER contained a statement that one of its subtheories was a copy of the ADDER theory. This, of course, is just one statement relating two theories. This section tries to illustrate more general cases of theories about theories which determine the large-scale structure of the program.

### 2.7.1 The THEORY Theory

The starting point is the theory of the prototypical theory and its construction. This theory simply reflects in logical language the data-structure definitions given earlier, with the simulation structure mentioning the procedures for accessing those structures. For example, the THEORY data-structure is reflected as the following.

```
In THEORY:
        Individual-variable T THEORY;
        Individual-variable L LANGUAGE;
        Individual-variable S SIMULATION-STRUCTURE;
        Individual-variable F FACTS;
        Individual-variable PARENT THEORY;
        Function-constant T-L (THEORY) (LANGUAGE);
        Function-constant T-S (THEORY) (SIMULATION-STRUCTURE);
        Function-constant T-F (THEORY) (FACTS);
        Function-constant T-P (THEORY) (THEORY);
        Axiom ∀T [∃L[L = T-P(T)] ∧ ∃S[S = T-S(T)] ∧ ∃F[F = T-F(T)] ∧ ∃PARENT[PARENT = T-P(T)]]
        Attach T-P (LAMBDA (X) (CXR 0 X));
        etc.
```

When we fill out this sort of theory, we obtain a complete description of the basic data-structures of the program, and the primitives for accessing, creating, and modifying them. I will not go into this here, for the full description is quite lengthy.[30]

### 2.7.2 Theories of Specific Theories

The THEORY theory only reflects the structure common to all logical theories. Other theories describe the structure common to all members of certain classes of theories. For example, the ADDER theory above describes a prototypical adder. If each of the components of this theory are reflected in the language of THEORY and related theories, we get a theory describing all theories copied from ADDER, containing, for instance

```
IN THEORY-OF-ADDER:
        Axiom: INDIVIDUAL-CONSTANT("A1", LANGUAGE(ADDER));
```

We can include these meta-theoretical statements in the theory itself, just as we include VC statements. Of course, we do not want to do this automatically for all statements, lest we reflect endlessly to produce an infinite number of such statements in each theory.

### 2.7.3 The VC Theory

The VC inference rule can be described by yet another theory, with contents like the following.[31]

```
IN VC:
        Individual-variable T1 THEORY;
        Individual-variable T2 THEORY;
        Individual-variable S1 WFF;
        Individual-variable S2 WFF;
        Axiom ∀t1 ∀t2 VC(T1, T2) ⊃
```

[30] Similar reflections can be made of the underlying Lisp system, by axiomatization of s-expressions and the primitives for creating and manipulating them. Weyhrauch and Cartwright and McCarthy [1979] have developed theories of Lisp along these lines.

[31] This is not quite correct or complete, as the exact details have yet to be worked out.

```
        ∀S1[S1 ∈ METATHEORY(T1) ⊃
              ∃S2[S2 ∈ METATHEORY(T2) ∧ S2 = SUBSTITUTION(S1)];
```

What this means is that to copy statements from one theory, one reflects the definition of the statement into a meta-theoretic statement, substitutes in the appropriate new names, adds the new meta-theoretical statement to the copy theory, and then de-reflects to get the copied object-level statement in the copy theory. Thus the definition of an individual constant S in the prototype would be reflected into a statement that S is an individual constant symbol. That meta-theoretical statement would be inferred in the copy theory, and de-reflected (treated as a definitional command) to realize S in the copy theory.

### 2.7.4 The PERSON Theory

Just as we progress from theories of things to theories of theories to theories of pairs of theories, we continue to theories describing the large-scale structure of the program as a theory of all currently existing theories. The abstract structure of the program we capture in the PERSON theory.

```
IN PERSON:
        Individual-constant THEORIES SET;
        Individual-constant BELIEFS SET;
        Individual-constant DESIRES SET;
        Individual-constant INTENTIONS SET;
        Individual-constant PROCEDURES SET;
        etc.
```

In addition, each of these parts of the program is attached to lists of concepts, beliefs, desires, intentions and procedures typical of all persons.[32] Of course, persons may be subclassified into types of persons, each of which has some extra or missing attitudes over those expected of persons in general. Further specializations lead to theories of particular persons, and then to theories of those persons in different temporal or hypothetical situations. We speculate on the use of these models of persons in hypothetical

[32] More likely, these sets are given only implicitly by predicates and procedures which recognize their extensions, and the typical contents are all listed in the tables of these procedures. The details of this have yet to be worked out.

reasoning and discourse in Chapter 7.

### 2.7.5 The Global Theory ME

> I am he as you are he as you are me and we are all together.
> John Lennon and Paul McCartney, *I am the Walrus*

The program itself is a theory, and this theory it describes as a modified copy of the PERSON theory. The program calls this theory of itself ME, and for simplicity, we will often do likewise, or alternatively use our name for the program, SEAN. That is, SEAN is our name for the program, not its name for itself, although it may know that others call it SEAN. The program refers to itself by containing the individual constant ME of type THEORY, and attaching itself to ME as ME's referent.[33]

ME is the parent theory, or context of existence, of all the program's theories, either directly or indirectly.[34] Thus ME's parent is ME, as is the parent of PERSON. This may seem paradoxical, to have a theory be a copy of one of its parts, but as we have constructed them, there is no inherent difficulty. In fact, PERSON ought to mention a ME symbol, but I have not worked out this detail.

ME also contains the symbol I, which serves as its name for its "self." I is normally attached to the global theory, that is, is coreferential with ME, but can be rebound to other person theories in hypothetical reasoning, as described in Chapter 7.

The normal operation of the program involves making changes to the theory denoted by I, that is, attachments are looked up in I, pathnames are interpreted in I, and inferences are made in I's theories.

[33] Weyhrauch uses the term META for this, but I don't for two reasons. First, the it is the system's theory of itself, for which the canonical term is "me" or "I", not "meta." Second, the term vulgarizes the memory of my paternal grandmother, Meta Enters Doyle, daughter of Hermann Enters.

[34] It seems possible in principle that the program might contemplate (but not employ) theories which have no parent. In fact, it might construct an entire other program in this way, or a description of another program, complete except for connections to the real world, and never running, because it can never get control. If the program then connects this other program to another processor, or sets up a time-sharing executive, it might have two minds running independently in the same machine, each with a different self.

## 2.8 Concepts and Attitudes

The preceding has explained how to create a hierarchically organized database of concepts to be used in representing things. But concepts are of little use unless they can be applied. This section indicates how concepts are used to form the attitudes of belief, desire, and intention which go to make up the mental state of the program.

The basic idea is simple. The global theory of the program contains statements about some of the concepts so as to create attitudes. By "attitudes" we mean "propositional attitudes" in the usual sense in which beliefs, wants, and intentions are propositional attitudes, and are viewed as a combination of an attitude and a propositional content. Thus "I want to eat some food" would be decomposed into the attitude "I want" and the propositional content "I eat some food", the combination notated as I-Want(I eat some food). This might be done in the program as follows. If **Raining** is a concept describing a state of affairs in which it is raining now, and if R1, R2, and R3 are all copies of Raining, then the global theory might contain statements BELIEF(R1), DESIRE(R2), and INTENTION(R3) to indicate its belief that it is raining, its desire that it be raining, or its intention that it be raining. We assume that one always makes particular instances or copies of concepts used in attitudes, just as one makes copies of concepts in forming parts of concepts. Thus, there might be commands Believed-concept, Desired-concept, and Intended-concept analogous to Typed-part, which automatically create the copy theory and whatever inference procedures (see below) are appropriate for relating the new concept to the current state of mind. This copying may be needless, but only further study can tell.

This realization of attitudes makes clear the distinction between the reasons for the concept involved and the reasons for attitudes involving the concept. That is, the program might have reasons for holding the concept theory in terms of the theories and procedures from which it was constructed. These reasons would have nothing to do with the reasons for the attitude statements in the global theory.

The global theory includes all the currently believed concepts as subtheories. That is, the program infers VC(ME, C) from BELIEF(C) in ME, so that all the statements in C are inferred as statements of ME. This scheme, or a variant using TYPED-PART, has considerable elegance, particularly when applied to plan concepts (as discussed in Section 4.9), in which the plan theory contains statements of several sorts of attitudes which are used to temporarily augment the current sets of attitudes for the duration of the plan. The exact details of this idea are yet to be resolved.

One important question is the relation between these concept-based attitudes with their reasons, and the logical statements and their reasons which go to make up concepts. There may well be a confusion of levels in my suggestions, as they seem to imply that attitudes (at least from one viewpoint) are really beliefs about attitudes, a conclusion raising many problems. Chapter 7 discusses this problem in more detail.

# CHAPTER 3

# FOUNDATIONS OF THE THEORY OF REASONING

"Ladder of wit! What madness is this?" Ebenezer demanded.

"No madness save the world's, sir. Take your wig question, now, that's such a thing in London: whether to wear a bob or a full-bottom peruke. Your simple tradesman hath no love for fashion and wears a bob on's natural hair the better to labor in; but give him ten pound and a fortnight to idle, he'll off to the shop for a great French shag and a ha'peck of powder, and think him the devil's own fellow! Then get ye a dozen such idlers; the sharpest among 'em will buy him a bob wig with lofty preachments on *the tyranny of fashion* -- haven't I heard 'em! -- and think him as far o'er his full-bottomed fellows as they o'er the merchants' sons and bob-haired 'prentices. Yet only climb a rung the higher, and it's back to the full-bottom, on a sage that's seen so many crop-wigs feigning sense, he knows 'tis but a pose of practicality and gets him a name for the cleverest of all by showing their sham to the light of day. But a grade o'er him is the bob again, on the pate of some philosopher, and over that the full-bottom, and so on. Or take your French question: the rustical wight is all for England and thinks each Frenchman the Devil himself, but a year in London and he'll sneer at the simple way his farm folk reason. Then comes a man who's traveled that road who says, 'Plague take this foppish shill-I, shall-I! When all's said and done 'tis England to the end!'; and after him your man that's been abroad and vows 'tis not a matter of *shill-I, shall-I* to one who's traveled, for no folk are cleverer than the clever French, 'gainst which your English townsman's but a bumpkin. Next yet's the man who's seen not France alone but every blessed province on the globe; he says 'tis the novice traveler sings such praise for Paris -- the man who's seen 'em all comes home to England and carries all's refinement in his heart. But then comes your grand skeptical philosopher, that will not grant right to either side; and after him a grander, that knows no side is right but takes sides anyway for the clever nonsense of't; and after him your worldly saint, that says he's past all talk of wars and kings fore'er, and gets him a great name for virtue. And after him --"

"Enough, I beg you!" Ebenezer cried, "My head spins! For God's sake what's your point?"

"No more than what I said before, sir: that de'il the bit ye've tramped about the world, and bleared your eyes with books, and honed your wits in clever company, whate'er ye *yea* is *nay'd* by the man just a wee bit simpler and again by the fellow just a wee bit brighter, so that clever folk care less for what ye think than why ye think it."

John Barth, *The Sot-Weed Factor*

In later chapters of this thesis, we discuss the question of which inferences to make, that is, how the reasoning process is controlled. We devote the present chapter to explaining the prior question of what we take inferences to be, and to describing the structure of a program based on this theory of reasoning.

## 3.1 The Nature of Reasoning

Reasoning involves changing one's attitudes from one set to another by adding some new attitudes and relinquishing others.[35] Reasoning includes not only "deductive" and "inductive" inferences, in which new beliefs are produced from prior beliefs via "deductive" and "inductive" rules of inference, but also "practical" inferences, in which new wants and intentions are produced from prior beliefs, wants, and intentions, and "changes of mind", in which one becomes unhappy with some belief or desire and discards it.

Reasoning is one sort of mental event, where by mental event I mean one's changing one's mind from one state or structural form to another. Reasoning, however, is not the only sort of mental event. For example, the creation of new mental data-structures which do not affect the set of attitudes is a non-reasoning mental event, as when one creates a new attachment or data-structure in SDL without giving it a justification.[36] Of course, most data-structures are created for use in changing the set of attitudes, but they need not all be of this form. For example, when a question arises concerning the truth of some proposition about which one has no opinion, one must first construct the proposition to be able to consider it. Only later, after one finds reasons for or against the proposition, does it enter the set of

[35] Harman [1973] develops the thesis that reasoning is a process of changing one's set of attitudes by adding some and abandoning others. Perhaps I misinterpret him, but I understand this to mean that one cannot have cases of reasoning which do not change the set of attitudes. Here, and later in this section, I propose a more general view, which incorporates such cases of reasoning.

Harman develops his view as part of his thesis that reasoning always increases the "explanatory coherence" of the set of attitudes. This view can be taken in at least two ways, either as a proposed control structure for the reasoning process, in which case the mechanisms I propose subsume and significantly extend this proposal, or as a proposal about what sorts of mental events count as cases of reasoning. But if this latter interpretation is his intent, his proposal seems to have serious flaws, of which I sketch three. A. It leaves out faulty reasoning, which is certainly reasoning, but need not always increase explanatory coherence. B. Harman's view either requires that explanatory coherence is a total order on the collection of sets of attitudes, which seems absurd, or that reasoning cannot involve changes of mind in which one switches from one "theoretical" interpretation of a set of "data" beliefs to another interpretation also explaining the "data" but incompatible with the original interpretation. This also seems unrealistic. C. I would think that there are many plans of reasoning which involve first decreasing explanatory coherence so as to later increase it, for example, making an assumption to see how it works out, reaching a paradox or contradiction, and then retracting the assumption to get a coherent set of beliefs.

The approach developed in this thesis, while motivated by rational thought, can also be used for some types of irrational thought. For example, the approach contains nothing that forces the program to avoid inconsistent intentions. Rather it is the values and procedures of the program which work to keep the set of intentions consistent. Similarly, the program can engage in rational thought even when it entertains conflicting beliefs. Indeed, to be able to think about how to escape its plight, it must be able to reason effectively in the presence of inconsistencies.

[36] In particular, the only unreasoned processes are those which (a) compute primitive justifications, (b) construct SDL data-structures prior to their justification, and (c) compute values to attach to constants in theories.

beliefs and thus directly into reasoning.

Although we must admit non-reasoning mental events such as the creation and destruction of data-structures, our aim will be to explain as many mental events as possible in terms of reasoning. We do not insist that all mental events always be performed by reasoning, just that it ought to be possible to perform any particular mental operation through reasoning when desired. This aim entails severe restrictions on the form of the program we adopt, restrictions on all aspects of program operation down to the basic processes of choosing and making inferences.

Why adopt such an aim? In rational actions one changes one's attitudes only for some reason, so a rational program should be able to explain its actions in terms of its reasons. If the program has explanations of its actions, then it can do many useful things, such as correct faulty rules of inference or beliefs, by examining and analyzing these explanations to trace effects to their sources.

But in this view, it appears, all mental events in a rational program would have a reason. Is this possible? Nearly so, as this thesis attempts to demonstrate. In later chapters we will manage to push just about everything into reasoning when necessary, from making inferences, to making choices, to taking actions on the basis of intentions.[37] Non-reasoning mental events will be used solely in the service of reasoning processes.[38]

The common view of reasoning differs from ours in taking reasoning to be the purely monotonic or additive process of adding new attitudes to the current set of attitudes, as in deductive inference. But that view has many inadequacies.

With this aim of embedding most of the program in reasoning when desired, we face the

[37] Even though they do not involve reasoning, the computations involved in non-reasoning mental processes can be introspected and analyzed for some purposes. We will discuss this further in the context of skill introspection.

[38] Other non-reasoning mental events include independent, non-destructive processes, such as the random creation of new data-structures, which do not hurt but may save work in later deliberate data-structure creation; sensory inputs, which will change independent of reasons due to causal connections to the world; and random destruction of data-structures, which is one (but only one) form of forgetting. Whether one wants to build random changes into one's rationality is still an unexplored question. Is there some utility in random events in thought, or are they just consequences of implementation in an imperfect, noisy machine? Note that even if one's mental processes involve no randomness, evolution would still involve random changes to the species as long as traditional reproductive methods remain the fashion.

problem that in the traditional view of reasoning, many changes of attitudes must be apparently non-reasoned, e.g. all non-monotonic or non-additive changes, all changes which do not increase the set of attitudes monotonically. One of the most important reasoning steps necessary for taking action is that of making predictions of the effects of the action. Making these predictions typically requires making assumptions about the current state of affairs, because one never knows everything relevant to the successful completion of an action. But once one has made such assumptions and predictions, one is invariably surprised on occasion, and finds the assumptions to have been incorrect, even though unavoidable. Then one has the problem of how to correct or revise one's beliefs so as to patch up one's beliefs in light of this new information. How can the theory of reasoning be formulated to accommodate these non-monotonic changes in the set of attitudes?

We answer this question by proposing a theory of reasoning in which all reasoning takes place by adding a record of an inference, called a *reason*, to the current set of reasons. Each reason is basically a record of an application of an inference rule or other procedure to some set of attitudes. The program then determines the current set of attitudes from this set of reasons by treating the set of reasons as the set of required inferences, as opposed to the merely possible inferences indicated by the inference rules themselves. That is, an inference rule indicates only potential constraints on the set of attitudes. Only after the inference rule has been applied to create actual inferences do those inferences constrain the current set of attitudes by means of the reasons recording the inferences. With this terminology, my thesis is as follows.

*Rational thought is a process of constructing reasons for attitudes.*

To say that some attitude (such as belief, desire, or intent) is rational is to say that there is some acceptable reason for holding that attitude. Rational thought is a process of finding such acceptable reasons.[39]

[39] Note that this thesis allows as rational thought inferences involving random choices. For example, we might count as an acceptable reason "I couldn't think of anything else to do, so I flipped a coin."

Whatever purposes the reasoner may have, such as solving problems, finding answers, or taking action, it operates by constructing reasons for believing things, desiring things, or intending things. The actual attitude in the reasoner occurs only as a by-product of constructing reasons. The current set of beliefs and desires arises from the current set of reasons for beliefs and desires, reasons phrased in terms of other beliefs and desires. When action is taken, it is because some reason for the action can be found in terms of the beliefs, desires, and intentions of the actor. I stress again, in this view the only *real* component of rational thought is the current set of reasons - the attitudes such as beliefs and desires arise from the set of reasons, and have no independent existence.

This view entails that for each possible attitude P just one of two states obtains: Either

(A) P has at least one currently acceptable (*valid*) reason, and is thus a member of the current set of attitudes, or

(B) P has no currently acceptable reasons (either no reasons at all, or only unacceptable ones), and is thus not a member of the current set of attitudes.

If P falls in state (A), we say that P is *in* (the current set of attitudes), and otherwise, that P is *out* (of the current set of attitudes). These states are not symmetric, for while reasons can be constructed to make P *in*, no reason can make P *out*. (If P is a belief, the most a new reason can do is to make $\neg$P *in* as well.)[40]

It would seem that the proposed view also succumbs to monotonicity problems, for the set of reasons grows monotonically, which (with the normal sense of "reason") leads to only monotonic increases in the set of current attitudes. To solve the problem of monotonicity, we introduce novel meanings for the terms "a reason" and "an assumption" in the context of belief attitudes. Similar theories apply to the other attitudes.

Traditionally, a reason for a belief consists of a set of other beliefs, such that if each of these basis beliefs is held, so also is the reasoned belief. To get off the ground, this analysis of reasons requires either circular arguments between beliefs (and the appropriate initial state of belief) or some fundamental type of belief which grounds all other arguments. The traditional view takes these fundamental beliefs, often called assumptions (or premises), as believed without reason. On this view, the reasoner makes changes in the the current set of beliefs by removing some of the current assumptions and adding some new ones.

To conform with the proposed view, we introduce meanings for "reason" and "assumption" such that assumptions also have reasons. A *reason* (technically, a SL-justification, as explained shortly)

[40] While this is a standard property of inference rules, it is not respected in the relatives of RMS developed by London [1978], McAllester [1978], and Thompson [1979]. In their systems, inferences are recorded as implications, not as inference steps. Thus if the program infers A from B, they record $A \supset B$, rather than $A \vdash B$. These two statements have different meanings. In their systems, if $A \supset B$ and $\neg B$ are both current beliefs, so also will be $\neg A$. But this violates the true meaning of the statement as a record of an inference, since if one has made the inference $A \vdash B$ and has $\neg B$, one need not be able to infer $\neg A$, since that ability depends on the inference rules defining $\vdash$. Even if $\vdash$ involves only the familiar inference rules, one cannot infer $\neg A$, but just that not $\vdash A$.

McAllester has defended his conflation of these notions on the grounds of the space efficiency of his program, that it simultaneously represent several justifications. but even if the semantic errors in his approach are ignored, it can be seen that the claimed space efficiency is an illusion stemming from an unrealistic assumption about the use of the program. Most propositions are used only in a positive form by the program, that is, it is the relatively rare proposition for which the program considers both the proposition and its opposite. This is so because most propositions are uncontroversial statements about the world or the structure and control of the program , rather than about questions being deliberated on. Thus RMS, which represents propositions and their opposites as distinct, unique data-structures. ultimately uses less space than McAllester's program, which represents propositions and their opposites as a separate CONS in each clause in which they occur.

The non-monotonic logic developed by McDermott and myself [1978] also appears to suffer from this confusion. There we suggested writing inference rules as implications, but I was never happy with this since it predicted somewhat different behavior from that of RMS. Reiter [1979] has since improved on this situation by developing a non-monotonic logic which properly treats justifications as inference rules, and thus avoids the problems with the earlier approach. It remains to be seen whether the modal approach McDermott and I develop can be reinterpreted or emended to avoid these confusions as well. McDermott [1980] strengthens the modal logic in an attempt at this. I would be very interested in a similar extension of the modal logic of provability in Peano arithmetic [Boolos 1979]. I would expect any correct provability-related logic to be an extension of that logic.

for a belief consists of an ordered pair of sets of other beliefs, such that the reasoned belief is *in* by virtue of this reason only if each belief in the first set is *in*, and each belief in the second set is *out*. An *assumption* is a current belief one of whose valid reasons depends on a non-current belief, that is, has a non-empty second set of antecedent beliefs. With these notions we can create "ungrounded" yet reasoned beliefs by making assumptions. (E.g. give P the reason ({},{¬P}).) We can also effect non-monotonic changes in the set of current beliefs by giving reasons for some of the *out* statements used in the reasons for current assumptions. (E.g. to get rid of P, justify ¬P.) We somewhat loosely say that when we justify some *out* belief supporting an assumption, (e.g. ¬P), we are *defeating*, *denying*, or *retracting* the assumption (P).

These new notions solve the monotonicity problem, thus overcoming the limitations of the traditional view of reasoning. Non-monotonic assumptions allow the program to make inferences with incomplete information about the actual state of affairs, and then to correct the conclusions drawn from these assumptions by later examining the set of reasons. We will give examples of this shortly.

Other advantages over the conventional view also follow. One of these advantages involves how the reasoner retracts assumptions. With the traditional notion of assumption, retracting assumptions was unreasoned. If the reasoner removed an assumption from the current set of beliefs, the assumption remained out until the reasoner specifically put it back into the set of current beliefs, even if changing circumstances obviated the value of removing this belief. The new notions introduce instead the *reasoned retraction of assumptions.* This means that the reasoner retracts an assumption only by giving a reason for why it should be retracted. If later this reason becomes invalid, then the retraction is no longer effective and the assumption is restored to the current set of beliefs.

The most important application of the reasoned retraction of assumptions is in dialectical argumentation, a technique we will employ extensively later in decision-making procedures. The basic idea is that one part of the program can put forward an argument for some conclusion based on some assumptions, where for this purpose we represent each of the steps of the argument as an assumption as well.[41] Other parts of the program wishing to disagree with the conclusion of the argument examine the argument to find some assumption or argument step they disagree with, and then present a new argument to defeat the chosen assumption or step. This new argument is constructed like the original one, so the original procedure or some other part of the program can try to defend the original conclusion by in turn defeating some assumption or step of the new argument with yet another argument. By adopting this representation for reasons uniformly, the program gains the ability to reflect on its inferences after the fact, and to simply not make the inferences if it decides it shouldn't have. If some step leads to paradox, the program need not make it, although the real progress will be made only if it further inquires into the reasons for its antecedents.

Records of inferences also help with the problem of determining the relevance of one belief to another. One can divide the problem of relevance into two parts: the more difficult one is the connection of one belief with another by some possible but yet unknown chain of inferences, the easier one is the connection of one belief with another by some past and recorded chain of inferences. Here we assume that any connections between beliefs stemming from their intended models are reflected in inference rules.

In this remainder of this chapter, we will describe the basis of our program organization by describing RMS, a program for recording reasons and revising beliefs. Further explanation of RMS can be found in [Doyle 1979]. RMS (Reason Maintenance System) renames and revises the TMS (Truth Maintenance System) presented in that paper. I changed the name not only because the program has nothing to do with truth, but also because the program is properly concerned with reasons for attitudes rather than the attitudes themselves.

In the remainder of this chapter, I describe RMS solely in terms of the attitude belief. In fact, RMS implements only a logic of belief, and not necessarily logics for any other attitudes. This results

[41] See Section 3.11 for the details of how this is done.

from a hypothesis and methodology I entertain but since have come to suspect, that the program can and should be designed so that it only uses beliefs, and embodies its intentions, say, in its beliefs about its intentions. Part of the motivation for this hypothesis comes from viewing the data-structures of SDL as statements in its meta-language, as mentioned in the previous chapter. The final chapter discusses possible problems with this approach, and possible solutions. For the time being, however, we accept this hypothesis and methodology, and pretend that the program works strictly with beliefs and beliefs about attitudes.

## 3.2 RMS, the Reason Maintenance System

RMS records and maintains arguments for potential program beliefs, so as to distinguish, at all times, the current set of program beliefs. It manipulates two data structures: *nodes*, which represent beliefs, and *justifications*, which represent reasons for beliefs. We write Content(N) to denote the statement of the potential belief represented by the node N. We say RMS believes in (the potential belief represented by) a node if it has an argument for the node and believes in the nodes involved in the argument. This may seem circular, but some nodes will have arguments which involve no other believed nodes, and so form the base step for the definition.

As its fundamental actions, (1) RMS can create a new node, to which the program attaches as its content a data-structure representing some belief. As mentioned in the previous chapter, the program attaches a RMS node to each of the data-structures representing the symbols of a language, the attachments of simulation structures, the facts in theories, etc. RMS performs no manipulation of the content of nodes. (2) It can add a new justification for a node, to represent a step of an argument for the belief represented by the node. This argument step represents the application of some inference rule or procedure. Inference rules and procedures all have RMS nodes and include these nodes in the justifications they create.[42] (3) Finally, RMS can mark a node as a *contradiction*, to represent the inconsistency of any set of beliefs which enter into an argument for the node. These markings will be used by RMS to signal the program whenever the marked node is brought *in*.

A new justification for a node may lead RMS to believe in the node. If did not believe in the node previously, this may in turn allow other nodes to be believed by previously existing but incomplete arguments. In this case, RMS invokes the *reason maintenance* procedure to make any necessary revisions in the set of beliefs. RMS revises the current set of beliefs by using the recorded justifications to compute non-circular arguments for nodes from premises and other special nodes, as described later. These non-circular arguments distinguish one justification as the *well-founded supporting justification* of each node representing a current belief. RMS locates the set of nodes to update by finding those nodes whose well-founded arguments depend on changed nodes.

RMS employs *non-monotonic justifications*, which, as explained previously, base an argument for a node not only on current belief in other nodes, as occurs in deductive inference, but also on lack of current belief in other nodes. For example, one might justify a node `N-1` representing a statement P on the basis of lack of belief in node `N-2` representing the statement ¬P. In this case, RMS would hold `N-1` as a current belief as long as `N-2` was not among the current beliefs, and we would say that it had assumed belief in `N-1`. More generally, by an *assumption* we mean any node whose well-founded support is a non-monotonic justification.

As a small example of the use of RMS, suppose that a hypothetical office scheduling program considers holding a meeting on Wednesday. To do this, the program assumes that the meeting is on Wednesday. The inference system of the program includes a rule which draws the conclusion that due to

[42] Actually, justifications mention not nodes but rather their contents. We do this so that it is easier to interpret the justifications when debugging the program, for otherwise one cannot easily read justifications to see what inference rules are involved, for one gets explanations like N-1 because N-2, N-3, and N-4, rather than B because Modus Ponens, A, and A⊃B. RMS always reads through the content data-structures to the RMS node involved via the function RMS-NODE. To make the exposition less complicated, all of the following is written as though the nodes themselves were mentioned in the justifications, rather than their contents.

regular commitments, any meeting on Wednesday must occur at 1:00 P.M. However, the fragment of the schedule for the week constructed so far has some activity scheduled for that time already, and so another rule concludes the meeting cannot be on Wednesday. We write these nodes and rule-constructed justifications as follows:

| *Node* | *Statement* | *Justification* | *Comment* |
|---|---|---|---|
| N-1 | DAY(M) = WEDNESDAY | (SL () (N-2)) | *an assumption* |
| N-2 | DAY(M) ≠ WEDNESDAY | | *no justification yet* |
| N-3 | TIME(M) = 13:00 | (SL (R-37 N-1) ()) | |

The above notation for the justifications indicates that they belong to the class of *support-list* (SL) justifications. Each of these justifications consists of two lists of nodes. A SL-justification is a *valid* reason for belief if and only if each of the nodes in the first list is believed and each of the nodes in the second list is not believed. In the example, if the two justifications listed above are the only existing justifications, then N-2 is not a current belief since it has no justifications at all. N-1 is believed since the justification for N-1 specifies that this node depends on the lack of belief in N-2. The justification for N-3 shows that N-3 depends on a (presumably believed) node R-37. In this case, R-37 represents a rule acting on (the statement represented by) N-1.

Subsequently another rule (represented by a node R-9) acts on beliefs about the day and time of some other engagement (represented by the nodes N-7 and N-8) to reject the assumption N-1.

| | | |
|---|---|---|
| N-2 | DAY(M) ≠ WEDNESDAY | (SL (R-9 N-7 N-8) ()) |

To accommodate this new justification, RMS will revise the current set of beliefs so that N-2 is believed, and N-1 and N-3 are not believed. It does this by tracing "upwards" from the node to be changed, N-2, to see that N-1 and N-3 ultimately depend on N-2. It then carefully examines the justifications of each of these nodes to see that N-2's justification is valid (so that N-2 is *in*). From this it follows that N-1's justification is invalid (so N-1 is *out*), and hence that N-3's justification is invalid (so N-3 is *out*).

### 3.3 RMS Data-structures

To make clear exactly what information is actually stored by RMS, as opposed to the information it computes on demand, this section presents the RMS data-structures. The following structure definitions in MIT Lisp Machine Lisp give the slots in the data-structures used to represent nodes and justifications. We have mentioned some of these already, and will explain many more in the following. Some, however, are for esoteric purposes not discussed here, but can be found in [Doyle 1979]. The structure presented here are simplified for clarity, as in the actual implementation some fields are full pointers, some are merely bits, and others are created only on demand.

```
(DEFSTRUCTURE NODE
  CONTENT                                         ;This chapter mentions these slots.
  SL-JUSTIFICATIONS
  CP-JUSTIFICATIONS
  SUPPORTING-JUSTIFICATIONS
  SUPPORTING-NODES
  CONSEQUENCES
  SUPPORT-STATUS
  CONTRADICTION-MARK
  NODE-MARK                                       ;These slots are not discussed.
  TMP-MARK
  NOTED-MARK
  FIS-MARK
  SUBORDINATES-MARK
  EXPLAIN-MARK
  SUPERIORS-MARK
  SIGNAL-RECALLING FUNCTION
  SIGNAL-FORGETTING-FUNCTION
  CP-CONSEQUENT-LIST)

(DEFSTRUCTURE SL-JUSTIFICATION
  INLIST
  OUTLIST)

(DEFSTRUCTURE CP-JUSTIFICATION
  CONSEQUENT
  INHYPOTHESES
  OUTHYPOTHESES)
```

### 3.4 States of Belief

A node may have several justifications, each justification representing a different reason for believing the node. These several justifications comprise the node's *justification-set.* The node is believed if and only if at least one of its justifications is *valid.* We described the conditions for validity of SL-justifications above, and shortly will introduce and explain the other type of justification used in RMS. We say that a node which has at least one valid justification is *in* (the current set of beliefs), and that a node with no valid justifications is *out* (of the current set of beliefs). We will alternatively say that each node has a *support-status* of either *in* or *out.* The distinction between *in* and *out* is not that between *true* and *false.* The former classification refers to current possession of valid reasons for belief. *True* and *false,* on the other hand, classify statements according to truth value independent of any reasons for belief.

In RMS, each potential belief to be used as a hypothesis or conclusion of an argument must be given its own distinct node. When uncertainty about some statement (e.g. P) exists, one must (eventually) provide nodes for both the statement and its negation. Either of these nodes can have or lack well-founded arguments, leading to a four-element belief set (similar to the belief set urged by Belnap [1976]) of neither P nor $\neg$P believed, exactly one believed, or both believed.

### 3.5 Justifications

Although natural arguments may use a wealth of types of argument steps or justifications, RMS forces one to fit all these into a common mold. RMS employs only two forms for justifications, called *support-list* (SL) and *conditional-proof* (CP) justifications. These are inspired by the typical forms of arguments in natural deduction inference systems.[43] Natural deduction is a sort of logical system in which there are no axioms, only inference rules. Proofs in natural deduction involve recording the steps

[43] See for example Suppes [1957].

of the proofs and the dependencies of each of these steps, that is, the set of hypotheses upon which each step depends. The inference rules then analyze the proof steps and dependencies to derive theorems which depend on no hypotheses. Two common inference rules are Modus Ponens and Discharging an Assumption. Modus Ponens is the familiar rule for detaching a conclusion from an implication and its antecedent. Discharging an Assumption is roughly the deduction theorem in action, which concludes an implication from the derivability of some statement from certain hypotheses, where the statement becomes the consequent of the implication and the hypotheses become the antecedents of the implication. These two inference rules respectively add and subtract dependencies from the support of a proof line. A proof in such a system might run as follows:

| *Line* | *Statement* | *Justification* | *Dependencies* | |
|---|---|---|---|---|
| 1. | $A \supset B$ | Premise | {1} | |
| 2. | $B \supset C$ | Premise | {2} | |
| 3. | $A$ | Hypothesis | {3} | |
| 4. | $B$ | MP 1,3 | {1,3} | |
| 5. | $C$ | MP 2,4 | {1,2,3} | |
| 6. | $A \supset C$ | Discharge 3,5 | {1,2} | |
| 7. | $A \supset B \wedge B \supset C$ | $\wedge$-introduction | | {1,2} |
| 8. | $(A \supset B \wedge B \supset C) \supset (A \supset C)$ | Discharge 7,6 | {} | *A Theorem* |

Each step of the proof has a line number, a statement, a justification, and a set of line numbers on which the statement depends. Premises and hypotheses depend on themselves, and other lines depend on the set of premises and hypotheses derived from their justifications. The above proof proves $A \supset C$ from the premises $A \supset B$ and $B \supset C$ by hypothesizing $A$ and concluding $C$ via two applications of Modus Ponens. The proof of $A \supset C$ ends by discharging the assumption $A$, which frees the conclusion of dependence on the hypothesis but leaves its dependence on the premises.

This example displays justifications which sum the dependencies of some of the referenced lines (as in line 4) and subtract the dependencies of some lines from those of other lines (as in line 6). The two types of justifications used in RMS account for these effects on dependencies. A support-list justification says that the justified node depends on each node in a set of other nodes, and in effect sums the dependencies of the referenced nodes. A conditional-proof justification says that the node it justifies depends on the validity of a certain hypothetical argument. As in the example above, it subtracts the dependencies of some nodes (the hypotheses of the hypothetical argument) from the dependencies of others (the conclusion of the hypothetical argument). Thus we might rewrite the example in terms of RMS justifications as follows (here ignoring the difference between premises and hypotheses, and ignoring the inference rule MP):

| | | | |
|---|---|---|---|
| N-1 | $A \supset B$ | (SL () ()) | *Premise* |
| N-2 | $B \supset C$ | (SL () ()) | *Premise* |
| N-3 | $A$ | (SL () ()) | *Premise* |
| N-4 | $B$ | (SL (N-1 N-3) ()) | *MP* |
| N-5 | $C$ | (SL (N-2 N-4) ()) | *MP* |
| N-6 | $A \supset C$ | (CP N-5 (N-3) ()) | *Discharge* |
| N-7 | $(A \supset B \wedge B \supset C) \supset (A \supset C)$ | (CP N-6 (N-1 N-2) ()) | *Discharge two assumptions* |

CP-justifications, which will be explained in greater detail below, differ from ordinary hypothetical arguments in that they use two lists of nodes as hypotheses, the *in*hypotheses and the *out*hypotheses. In the above justification for N-6, the list of *in*hypotheses contains just N-3, and the list of *out*hypotheses is empty. This difference results from our use of non-monotonic justifications, in which arguments for nodes can be based both on *in* and *out* nodes.

### 3.6 Support-list Justifications

To repeat the definition scattered throughout the previous discussion, the support-list justification has the form

(SL <*in*list> <*out*list>),

and is valid if and only if each node in its *in*list is *in*, and each node in its *out*list is *out*. The

SL-justification form can represent several types of deductions. With empty *in*list and empty *out*list, we say the justification forms a *premise* justification. A premise justification is always valid, and so the node it justifies will always be *in*. SL-justifications with nonempty *in*lists and empty *out*lists represent normal deductive inferences. Each such justification represents a monotonic argument for the node it justifies from the nodes of its *in*list. We define *assumptions* to be nodes whose supporting-justification has a nonempty *out*list. These assumption justifications can be interpreted by viewing the nodes of the *in*list as comprising the reasons for wanting to assume the justified node; the nodes of the *out*list represent the specific criteria authorizing this assumption. For example, the reason for wanting to assume "The weather will be nice" might be "Be optimistic about the weather"; and the assumption might be authorized by having no reason to believe "The weather will be bad." We occasionally interpret the nodes of the *out*list as "denials" of the justified node, beliefs which imply the negation of the belief represented by the justified node.

To make the exposition less jargonistic, we occasionally use the phrases "N-1 is justified (non-)monotonically in terms of N-2" and "N-1's justification (non-)monotonically involves N-2" to mean that N-2 occurs in the *in*list (*out*list) of N-1's justification.

### 3.7 Terminology of Dependency Relationships

I must pause to present some terminology before explaining CP-justifications. The definitions of dependency relationships introduced in this section are numerous, and the reader should consult Figures 5, 6, and 7 for examples of the definitions.

As mentioned previously, RMS singles out one justification, called the *supporting-justification*, in the justification-set of each *in* node to form part of the non-circular argument for the node. For reasons explained shortly, all nodes have only SL-justifications as their supporting-justifications, never CP-justifications. The set of *supporting-nodes* of a node is the set of nodes which RMS used to determine

| Node | Justification | Justification Name |
|---|---|---|
| 1 | (SL (6) ()) | J1 |
| 2 | (SL () (1)) | J2 |
| 3 | [illegible] | J3 |
| 4 | (SL (2) ()) | J4A |
| 4 | (SL (3) ()) | J4B |
| 5 | (SL () ()) | J5 |
| 6 | (SL (3 5) ()) | J6 |

Six Nodes and Seven Justifications

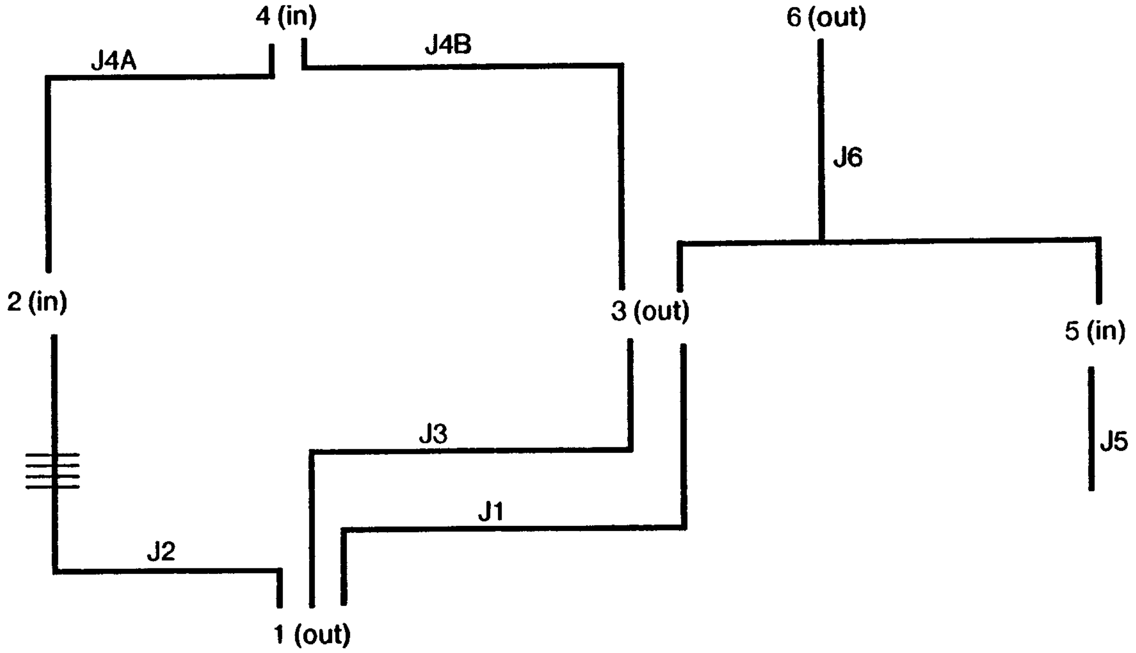


Figure 6

A depiction of the previous system of justifications and nodes.
All arrows represent justifications. The uncrossed arrows represent inlist, and only the crossed line of J2 represents an outlist.
We always visualize support relationships as pointing upwards.

| Dependency | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 | Node 6 |
|---|---|---|---|---|---|---|
| Support-status | out | in | out | in | in | out |
| Supporting-justification | - | J2 | - | J4A | J5 | - |
| Supporting-nodes | 3 | 1 | 1 | 2 | - | 3 |
| Antecedents | - | 1 | - | 2 | - | - |
| Foundations | - | 1 | - | 1,2 | - | - |
| Ancestors | 1,3 | 1,3 | 1,3 | 1,2,3 | - | 1,3 |
| Consequences | 2,3 | 4 | 1,4,6 | - | 6 | - |
| Affected-consequences | 2,3 | 4 | 1,6 | - | - | - |
| Believed-consequences | 2 | 4 | - | - | - | - |
| Repercussions | 1,2,3,4,6 | 4 | 1,2,3,4,6 | - | - | - |
| Believed-repercussions | 2,4 | 4 | - | - | - | - |

Figure 7

A table of all the dependency relationships implicit in the system of justifications. Dashed entries are empty. All other entries are lists of nodes in the dependency relationship to the node at the top of the column.

the support-status of the node. For *in* nodes, the supporting-nodes are just the nodes listed in the *in*list and *out*list of its supporting-justification, and in this case we also call the supporting-nodes the *antecedents* of the node. For the supporting-nodes of *out* nodes, RMS picks one node from each justification in the justification-set. From SL-justifications, it picks either an *out* node from the *in*list or an *in* node from the *out*list. From CP-justifications, it picks either an *out* node from the *in*hypotheses or consequent or an *in* from the *out*hypotheses. We define the supporting-nodes of *out* nodes in this way so that the support-status of the node in question cannot change without either a change in the support-status of one of the supporting-nodes, or without the addition of a new valid justification. We say that an *out* node has no antecedents. RMS keeps the supporting-nodes of each node as part of the node data-structure, and computes the antecedents of the node from this list.

The set of *foundations* of a node is the transitive closure of the antecedents of the node, that is, the antecedents of the node, their antecedents, and so on. This set is the set of nodes involved in the well-founded argument for belief in the node. The set of *ancestors* of a node, analogously, is the transitive closure of the supporting-nodes of the node, that is, the supporting-nodes of the node, their supporting-nodes, and so on. This set is the set of nodes which might possibly affect the support-status of the node. The ancestors of a node may include the node itself, for the closure of the supporting-nodes relation need not be well-founded. RMS computes these dependency relationships from the supporting-nodes and antecedents of nodes.

In the other direction, the set of *consequences* of a node is the set of all nodes which mention the node in one of the justifications in their justification-set. The *affected-consequences* of a node are just those consequences of the node which contain the node in their set of supporting-nodes. The *believed-consequences* of a node are just those *in* consequences of the node which contain the node in their set of antecedents. RMS keeps the consequences of each node as part of the node data-structure, and computes the affected- and believed-consequences from the consequences.

The set of *repercussions* of a node is the transitive closure of the affected-consequences of the node, that is, the affected-consequences of the node, their affected-consequences, and so on. The set of *believed-repercussions* of a node is the transitive closure of the believed-consequences of the node, that is, the believed-consequences of the node, their believed-consequences, and so on. RMS computes all these relationships from the consequences of the node.

In all of the following, I visualize the lines of support for nodes as directed upwards, so that I look up to see repercussions, and down to see foundations. I say that one node is of lower level than another if its believed-repercussions include the other node.

### 3.8 Conditional-proof Justifications

With this terminology, we can now begin to explain conditional-proof justifications. The exact meaning of these justifications in RMS is complex and difficult to describe, so the reader may find this section hard going. CP-justifications take the form

`(CP <consequent> <`*in*`hypotheses> <`*out*`hypotheses>).`

A CP-justification is valid if the consequent node is *in* whenever (a) each node of the *in*hypotheses is *in* and (b) each node of the *out*hypotheses is *out*. Except in a few esoteric uses described later, the set of *out*hypotheses is empty, so normally a node justified with a CP-justification represents the implication whose antecedents are the *in*hypotheses and whose consequent is the consequent of the CP-justification. Standard conditional-proofs in natural deduction systems typically specify a single set of hypotheses, which corresponds to the *in*hypotheses of a CP-justification. In the present case, the set of hypotheses must be divided into two disjoint subsets, since nodes may be derived both from some nodes being *in* and other nodes being *out*. Some deduction systems also employ multiple-consequent conditional-proofs. We forego these for reasons of implementation efficiency.

RMS handles CP-justifications in special ways. It can easily determine the validity of a CP-justification only when the justification's consequent and *in*hypotheses are *in* and the *out*hypotheses are *out*, since determining the justification's validity with other support-statuses for these nodes may require switching the support-statuses of the hypothesis nodes and their repercussions to set up the hypothetical situation in which the validity of the conditional-proof can be evaluated. This may may require reason maintenance processing, which in turn may require validity checking of further CP-justifications, and so the whole process becomes extremely complex. Instead of attempting such a detailed analysis (for which I know no algorithms), RMS uses the opportunistic and approximate strategy of computing SL-justifications currently equivalent to CP-justifications. At the time of their creation, these new SL-justifications are equivalent to the CP-justifications in terms of the dependencies they specify, and are easily checked for validity. Whenever RMS finds a CP-justification valid, it computes an equivalent SL-justification by analyzing the well-founded argument for the consequent node of the CP-justification to find those nodes which are not themselves supported by any of the *in*hypotheses or *out*hypotheses but which directly enter into the argument for the consequent node along with the hypotheses. Precisely, RMS finds all nodes N in the foundations of the consequent such that N is not one of the hypotheses or one of their repercussions, and N is either an antecedent of the consequent or an antecedent of some other node in the repercussions of the hypotheses. The *in* nodes in this set form the *in*list of the equivalent SL-justification, and the *out* nodes of the set form the *out*list of the equivalent SL-justification. RMS attaches the list of SL-justifications computed in this way to their parent CP-justifications, and always prefers to use these SL-justifications in its processing. RMS checks the derived SL-justifications first in determining the support-status of a node, and uses them in explanations. It uses only SL-justifications (derived or otherwise) as supporting-justifications of nodes. The accuracy and limitations of this approximation are open problems.

### 3.9 Circular Arguments

Suppose a program manipulates three nodes as follows:

| | | |
|---|---|---|
| *F* | (= (+ X Y) 4) | *omitted but valid* |
| *G* | (= X 1) | (SL (*J*) ()) |
| *H* | (= Y 3) | (SL (*K*) ()). |

If *J* is *in* and *K* is *out*, then RMS will make *F* and *G in*, and *H out*. If the program then justifies *H* with

(SL (*FG*) ()),

RMS will bring *H in*. Suppose now that RMS makes *J out* and *K in*, leading to *G* becoming *out* and *H* remaining *in*. The program might then justify *G* with

(SL (*FH*) ()).

If RMS now takes *K out*, the original justification supporting belief in *H* becomes invalid, leading RMS to reassess the grounds for belief in *H*. If it makes its decision to believe a node on the basis of a simple evaluation of each of the justifications of the node, then it will leave both *G* and *H in*, since the two most recently added justifications form circular arguments for *G* and *H* in terms of each other.

These circular arguments supporting belief in nodes motivate the use of well-founded supporting justifications, since nodes imprudently believed on tenuous circular bases can lead to ill-considered actions, wasted data base searches, and illusory inconsistencies which might never have occurred without the misleading, circularly supported beliefs. In view of this problem, the algorithms of RMS must ensure that it believes no node for circular reasons.

Purported arguments for nodes can contain essentially three different kinds of circularities, each of which must be handled in a different way. The first and most common type of circularity involves only nodes which can be taken to be *out* consistently with their justifications. Such circularities arise routinely through equivalent or conditionally equivalent beliefs and mutually constraining beliefs. The above algebra example falls into this class of circularity. In this case, RMS makes all of the involved nodes *out*.

The second type of circularity includes at least one node which must be *in*. Consider, for example

```
F      TO-BE        (SL () (G))
G      ¬TO-BE       (SL () (F)).
```

In the absence of other justifications, these justifications force RMS either to make *F in* and *G out*, or *G in* and *F out*. When RMS meets such a circularity, it must choose some one of these nodes *in*. This decision frequently affects the actions of the program drastically, so it must often be made carefully using the revision techniques outlined below.

In unsatisfiable circularities, the third type, no assignment of *in* or *out* to nodes is consistent with their justifications. Consider

```
F      ...          (SL () (F)).
```

With no other justifications for *F*, RMS must make *F in* if and only if it makes *F out*, an impossible task. Unsatisfiable circularities sometimes indicate real inconsistencies in the beliefs of the program using the reason maintenance system. If so, RMS must discard one of the justifications involved.[44]

### 3.10 The Reason Maintenance Process

The reason maintenance process makes any necessary revisions in the current set of beliefs when the program adds to the justification-set of a node. We only outline it here. For more detail, see [Doyle 1979].

The reason maintenance process starts when a new justification is added to a node. Only minor

[44] Discarding a justification violates the thesis of rationality proposed earlier. However, as Section 3.11 explains, the program always employs defeasible justifications, so unsatisfiable circularities never arise. This saves the thesis of rationality and allows explanations of the revision as well.

bookkeeping is required if the new justification is invalid, or if it is valid but the node is already *in*. If the justification is valid and the node is *out*, then the node and its repercussions must be updated. RMS makes a list containing the node and its repercussions, and marks each of these nodes to indicate that they have not yet been given well-founded support. RMS then examines the justifications of these nodes to see if any are valid purely on the basis of unmarked nodes, that is, purely on the basis of nodes which do have well-founded support. If it finds any, these nodes are brought *in* (or *out* if all their justifications are invalid purely on the basis of well-founded nodes). Then the marked consequences of the nodes are examined to see if they too can now be given well-founded support. Sometimes, after all of the marked nodes have been examined in this way, well-founded support-statuses will have been found for all nodes. Sometimes, however, some nodes will remain marked due to circularities. If so, RMS constructs a decision intention to decide between revisions, so that the decision about which belief revision to use may be made carefully if desired. Otherwise, the default decision is to choose a revision randomly by a constraint-relaxation process which assigns support-statuses to the remaining nodes. The new intention does not depend on any prior beliefs, in particular not on the beliefs under revision, so its addition does not invoke another revision decision.

If the revision decision is made carefully, it involves analyzing the circularity to see what the alternative revisions are. This analysis can be very involved, and we have not pursued it very extensively. One early version of RMS [Doyle 1976] applied graph-theoretic algorithms to first analyze the circularity into strongly connected components, and then to sort these components topologically.[45] The minimal (in the sort order) strongly connected components are the obvious candidates for closer examination, as non-minimal components cannot be decided without first deciding the minimal components. The early RMS would then pick (randomly) one node from each minimal component to be *out*, and determine the

[45] Later versions of RMS abandoned this technique because it was unnecessarily complicated for the small belief systems being manipulated, and since it involved answering a number of questions which involve considerable study in the context of large complicated belief systems. It would be nice if someone would take up this problem again and explore it carefully.

statuses of the other nodes in the component from this new constraint. After all the repercussions of these choices had been accounted for, it would repeat this process of analysis and choice until the statuses of all beliefs had been settled. This is a very tricky procedure, for these choices of revision might be wrong, and so lead to apparently unsatisfiable inconsistencies. To avoid this, it appears that the decision should involve adding a new justification to the chosen nodes to set their status, rather than just setting it arbitrarily.

While processing the repercussions of a decision, RMS can detect an apparently unsatisfiable circularity and again invoke a decision intention to either change one of the previous decisions or, as a last resort, discard one of the justifications involved in the unsatisfiable circularity.

RMS also handles contradictions using this technique. Whenever it brings a node *in* that has been marked as a contradiction, RMS constructs a new intention whose aim is to resolve the inconsistency.

Actually, the existing RMS (TMS) has not been altered to provide for careful selection of revisions by intentions, but just makes the revisions randomly. I do not see any overwhelming difficulties in carrying through all these alterations. In any event, even if the current version is used, the net result will be that the program will be a less efficient than it might otherwise be. These changes can always be put in later.

### 3.11 Defeasible Reasons and Dialectical Argumentation

> Everything and everyone has to be criticized if there is to be any progress in the world. Anybody ought to be prepared for that and grant everyone else that right.
> H. Enters [1924, p. 100]

We have just described the basics of RMS, but the program uses RMS in a special way. In the above we described only ways for RMS to update the current set of beliefs by adding new justifications. We made no provisions for removing justifications, for we wish to make all changes in beliefs for good reasons. To allow all justifications to be defeasible, we reflect all justifications in explicit program beliefs about the justifications, and make all these beliefs assumptions.

Suppose the program wants to justify node $N$ with the justification (SL $I$ $O$). Instead of doing this directly, it creates a new node, $J$, representing the statement that $I$ and $O$ SL-justify $N$; in other words, that belief in each node of $I$ and lack of belief in each node of $O$ constitute a reason for believing in $N$. The program justifies $N$ with the justification (SL $J+I$ $O$), where $J+I$ represents the list $I$ augmented by $J$. RMS will make $N$ *in* by reason of this justification only if $J$ is *in*. The program also creates another new node, $\neg J$, representing the statement that $J$ represents a challenged justification. It then justifies $J$ with the justification (SL () ($\neg J$)). Note that this justification is not reflected in a corresponding belief, but is a simple justification.[46] In this way, the program makes a new node to represent the justification as an explicit belief, and then assumes that the justification has not been challenged.

To do this, the program never directly calls the functions RMS-SL-JUSTIFY and RMS-CP-JUSTIFY which create basic RMS justifications for nodes. Instead, it calls `(SL-JUSTIFY node inlist outlist)` and `(CP-JUSTIFY node consequent inhypotheses outhypotheses)`. What SL-JUSTIFY does (and CP-JUSTIFY analogously), is to create a new individual constant in ME of the form J-nnn and then use this as the name of a new fact in ME whose wff is

```
SL-JUSTIFICATION(J-nnn, node, {J-nnn}+inlist, outlist).
```

SL-JUSTIFY also creates a new fact in ME called D-nnn, whose wff is

```
DEFEATED(J-nnn).
```

It then creates two basic RMS justifications. As the first, it justifies J-nnn with an empty *in*list and an

[46] There is a slight modification of this technique which avoids having these non-reflected justifications. To do this, we do not create the new node $J$, but only $\neg J$, and make the actual justification be ( SL $I$ $O+\neg J$). The net effect is the same.

*out*list containing just D-nnn. It then justifies `node` with the *in*list containing both J-nnn and the original *in*list, and the original *out*list.

For example, suppose the program wishes to conclude that the value of Y is 1 in a theory T-1 in which the value of X is 3 and the known that X and Y sum to 4.

```
IN T-1:
N-1     Individual-constant X;          Justifications omitted
N-2     Individual-constant Y;
N-3     Individual-constant 4;
N-4     Function-constant +;
N-5     Attach X 3;
N-6     Attach 4 4;
N-7     Attach + +;
N-8     Axiom X + Y = 4
N-9     Attach Y 1;                     No justification yet
```

It would first switch to ME to describe the justification of the new attachment of Y to 1 as describe above, and then switch back to T-1 to make the actual justification.

```
IN ME:
N-10    Fact SL-JUSTIFICATION(N-10, N-9, {N-10 N-1 ... N-9}, {});
                                        (SL () (N-11))
N-11    Fact DEFEATED(N-10)             No justification yet
IN T-1:
N-9     Attach Y 1;                     (SL (N-10 N-1 ... N-9) ())
```

If the program then wished to defeat this justification, it would again go to ME and construct the following.

```
IN ME:
N-12    SL-JUSTIFICATION(N-12, N-11, {N-12, ...} {...});
                                        (SL () (N-13))
N-13    DEFEATED(N-12);                 No justification yet
N-11    DEFEATED(N-10);                 (SL (N-12 ...) (...))
```

This organization of the reasoning process into dialectical argumentation has three interesting aspects. The first is that any belief of the program may be abandoned, since the program only believes for reason, and all reasons can be reconsidered and rejected after the fact.

Second, RMS no longer has to worry about truly unsatisfiable circularities. Since all assumptions are defeated by reasons which are themselves assumptions, what in the direct use of RMS would be unsatisfiable circularities are in this indirect use just defeasible reasons. Thus RMS never needs to discard a justification. It only has to defeat the justification with another.

Third, this organization clarifies the meaning of CP-justifications. It shows that CP-justifications actually compute arguments. Suppose the program draws the conclusion C from A and B via the justification J. If the program justifies D with `(CP C (A B) ())`, it is justifying D on the grounds that an argument exists for C from A and B, but that argument is just J! This new justification then is equivalent in this case to `(SL (J) ())`. Thus the CP-justification in effect returns an argument of one belief as the support (as an object) for another belief.

This concludes the discussion of the underlying reasoning framework. We now turn to the means by which reasoning is controlled and applied in its own service.

# CHAPTER 4

# DELIBERATE ACTION

*De l'audace, et encore de l'audace, et tojours de l'audace!*
Georges Jacques Danton

One of the most important things about man is his ability to adapt so as to further his survival. But to adapt, man changes both his environment and himself, body and mind. To do this, however, he must be aware of himself and his environment.

But awareness is sometimes difficult to attain. For example, it is usually difficult to fully grasp the effects of one's actions. One contracts an outsider to build one's home only to discover that the social benefits of communal home-raising have been lost. One builds a dam to assure regular crops and discovers the destruction of wilderness upstream and wildlife downstream. One paints one's nails only to discover them cracking in inconvenient moments. One selects one's children's genes to avoid hemophilia and caries to discover unexpected diabetes. And one finally learns how to concentrate well on one's work to succeed, only to appear distant and uncaring to one's family and companions. Man may not always have all the information he needs to act successfully, but he must always be concerned with the direction of change, and to try to control that direction as best he can.

To control the direction of change, man needs to be conscious of the current state of affairs and the desired state of affairs, and of the effects of various actions he might take, conscious of his surroundings, his body, and his mind. AI studies of problem solving have touched on many ways of problem solving and planning, but typically these are applied not to all objects of change, not to the program's own mental state, but only to external objects such as chess games, housebuilding, or electronic circuits. Psychologists and popular writers have not neglected mental change, as an enormous self-help literature attests.[47] But in spite of this, AI seems to have largely rested content with attacking the physical, not the mental, problem domains. Because of this, I believe, self-consciousness, which humans frequently think of as their hallmark and gift over the other animals, has been viewed as a mystical topic, something for future generations of scientists to conquer.

As I hope to indicate in this and later chapters, self-consciousness is no mystic apparition, but a practical device to be readily applied to controlling reasoning. Self-consciousness is easy to achieve, as long as one is not blinded by an overriding preoccupation with physical affairs.

This chapter lays out the basics of how a program can be conscious of and reflect on its own plans, intentions, actions, reasons, decisions, and beliefs. The following chapters study decision-making, modifying beliefs, and modifying skills as deliberate, conscious activities.

To be able to tell what one is doing is crucial for making plans, making decisions, and learning. One can hardly make plans to achieve one's desires if one cannot tell what one wants.[48] Rational decisions are sometimes described as those which "fit best" with one's beliefs, desires, and intentions, so to make rational decisions one needs to take one's intentions into account. One can hardly help painting oneself into a corner unless one neglects one's intention to leave the room after the job is done. And when learning, one cannot assign credit or blame to one's beliefs or procedures unless one can explain what one did and why, that is, one's actions, intentions, and reasons.[49] Thus for one's own benefit in planning, in evaluating one's successes, and in modifying one's beliefs and skills, one needs to be able to distinguish which effects of one's actions are intentional and which are unintentional, since one can always hope to correct unintentional bad effects.

[47] For example, see [Russell 1930], [Carnegie 1936, 1944], [Ellis and Harper 1961], or [Johnson 1977]. Johnson's book gives an explicit (and to AI folk, familiar) problem solving procedure for changing one's skills and attitudes towards the world: reducing problems to subproblems, monitoring their progress, etc., all towards ends like becoming good at carrying on conversations and learning to tolerate or accept one's current limitations.

[48] This is not to say that desires cannot influence one's behavior unless conscious. Freudian psychoanalysis goes to great lengths to ferret out unconscious desires. We do not pursue here exactly how such unconscious attitudes might be realized.

[49] As mentioned earlier, our use of the term "reason" refers to inference records, not to antecedents. The reader is cautioned that much of the philosophical literature on action uses yet another meaning for "reason," namely desires, motives, or volitions underlying actions.

The basic problems discussed in this chapter are how a program can tell what it is doing and how it can act on its intentions. The chapter discusses in turn the library of plans, the constituents of plans, the current state of mind, and the interpreter.

## 4.1 Plan Generation, Execution, and Interpretation

Traditional approaches to the construction of complex patterns of actions rely on a distinction between plan generation and execution. In that view, the reasoner takes a problem description and constructs a sequence of instructions for the action mechanism of a machine. The reasoner constructs this sequence of commands so that it believes their execution will solve the problem it accepted. Once the command sequence has been constructed, it is given to the action mechanism, which carries out each of the instructions.

We do not adopt this two-stage approach, for it has several drawbacks. One problem is that it makes error handling largely a matter of foresight. Actions of all kinds are notoriously prone to failures in unusual circumstances. To make an effective plan that is guaranteed to work is impossible. There are always circumstances in which a particular plan will fail to realize its intended effect. The best the plan generator can do is to try to build the sequence of instructions so that it incorporates conditional steps which handle all of the possible failures that might arise. But this is a poor strategy, both because there are in general a huge number of forseeable difficulties, and because there are in general always unforeseen but possible difficulties. For similar reasons, the separation of plan generation and execution makes information gathering steps awkward to plan, for these are steps which explicitly have many possible (and perhaps unpredictable) outcomes. Thus this separation of plan generation and execution is untenable. The reasoner must always be ready to replan the necessary steps whenever a plan fails. For example, STRIPS [Fikes and Nilsson 1971] would devise a plan to be executed by PLANEX [Fikes 1972], which would reinvoke STRIPS to replan whenever actions failed. STRIPS could not produce conditional plans, so this was its only possible recourse.

In our case, there is yet another reason against dividing these processes. When this division is made, it makes impossible the planning of the reasoning involved in the generation process. Since we view reasoning as a species of action, we cannot construct a plan without taking actions themselves requiring planning, and we cannot wait until the plan is constructed before executing it, for otherwise reasoning actions can never occur.

The most natural strategy, and the one we adopt, is to mix plan generation and execution in a process better described as self-interpretation. This consists of repeatedly acting on one's intentions, many of which involve the formation of intentions for further actions. Thus error handling and information gathering steps (of which inferential reasoning steps can be viewed as an important subclass) are handled by forming intentions to carry out the step and then reflecting on the result, where the reflection involves the same reasoning processes which went into the formation of the step itself.

The interpretation organization of the reasoner avoids the ill-considered separation of plan generation and execution, making the normal activity one of reasoning about how to take the next reasoning steps, which themselves repeat this activity, so that the reasoner is constantly reasoning about how to reason. The basic steps of the program's operation are (1) to examine the set of desires to possibly decide to pursue some of them, that is, to form some new intentions, (2) to examine the current set of intentions to select one to work on next, (3) to examine the library of procedures to select some way of carrying out the intention, (4) to carry out the selected intention by executing the selected procedure if it is a primitive, and by adding it to the current state of mind if it is a plan, and (5) to repeat these steps.

The next section describes the library of procedures, which contains the primitives and plans for both external and internal reasoning and other actions.

## 4.2 Plans and the Library of Procedures

Plans are ways of describing the structure of one's desires and intentions. Plans, as I use the term, are complex concepts made up of many other sorts of concepts, including desires, intentions, and subplans. We will describe these sorts of concepts in more detail below, along with the other sorts of information that go to make up plans.

Plans play an important role in the operation of the program, and are stored in the *procedure library* (also called the *plan library*).[50] The procedure library lists all procedures of the program as attachments between the procedure's name and the procedure itself. It also contains a number of sorts of statements about the procedures, but we will discuss those later. Thus plans and primitives contain the "how-to" information of the program. The "know-how" of the program results from combining plans and primitives with information about their use, such as indexing them by their important effects.

Plans differ greatly in their specificity. The plan library typically will contain very general plans useful when one has nothing better to try. These general plans include the standard problem solving techniques, the "weak methods" as Newell terms them [Newell 1969]. But plans can be specific as well. The typical procedure library also includes plans for specific tasks, such as (depending on the domains of expertise of the program) how to design Butterworth filters, how to build a three-bedroom Colonial house in the northeast, how to make airplane reservations, and how to make cheesecake.

This notion of specificity can be factored into two sorts of specificity. Part of the context of the plan can be stated in the sort of problem the plan is applicable to, and part of the context can be stated in restrictions on when the plan is considered defined. In terms that we will explain in more detail below, this just means that the context of applicability of the plan can be stated in both the justification of the plan and in the indexing of the plan by its relevant effects. For example, consider a plan for putting out a

[50] Actually, the procedure library is a fiction just like the sets of beliefs, desires, and intentions. Procedures are each concepts, and thus are a subset of the concepts of the program, but distinguished as procedures by statements about them in the global theory, either `PLAN(concept)` or `PRIMITIVE(concept)`.

grease fire when cooking. This plan has crucial differences from the ordinary procedure for putting out a fire, namely that one should not use water as a suffocant. The question of factoring the context of plan specificity can be seen here in the following suggestions. One can index the plan under the problem of how to put out a fire while cooking with grease, or one can instead have one's plan for cooking with grease temporarily define a new plan for putting out fires and temporarily mask the usual plan for putting out fires with the new one.

This separation of the context of applicability of plans into relevancy and definitional components may seem unimportant, but I think it bears a message not to be neglected in the design of the program. If one only uses relevancy indexing, which is standard in most traditional AI programs, one is forced to face severe runtime retrieval problems. On the other hand, the combined approach allows one to do a good bit of work when setting up the problem to be attacked. If the problem is complex, then there will be many considerations necessary in judging whether a plan is relevant to a subproblem, and so the retrieval problem will be very great. If one knows beforehand that the problem is complex, one is willing to spend a good bit of time on preparing for the execution of the plan. This can be seen in the standard human practices in which people who perform complex tasks are give training or manuals to read specifying the procedures to use when special circumstance arise. Someone may have a great talent in looking up what to do in reference sources, but he will not be employed in many complex tasks on this basis. One can hardly expect a soldier in the field to continually look up procedures for what to do about his problems.

For these reasons, our plans are not simply composed of a few goals and temporal ordering relationships between them, as is common in many other AI systems which use the term "plan". Our plans contain not only these things, but also beliefs to be held as assumptions while carrying out the plan, locally defined plans for handling foreseen special cases, and guidelines for making the decisions expected to be encountered while carrying out the plan. In this way it is more appropriate to view plans as specifying partial states of mind or sets of attitudes to adopt for the duration of the execution of the plan.[51] In this view, the current state of mind is the sum of a set of realized plans, so that plans reproduce in the small the structure of the program in the large.

We represent plans as theories in SDL, and when instances of plans are added to the current state of mind, versions of all the terms, attachments, and statements in the theory are added to the theory representing the current state of mind. Thus, the description of the form of plans is largely a matter of describing the sorts of things plans can contain.

## 4.3 The Ambiguous "Goal"

Before proceeding, we first digress to point out a long-standing confusion in artificial intelligence, and perhaps in psychology and philosophy as well. The term "goal" in common technical usage seems to have no fixed meaning. It seems instead to be used on different occasions to mean both "desire" and "intention". I have searched many places, and no where do I find any discussions explaining what "goal" is supposed to mean, or how it relates to the less technical notions of desire and intent. This may seem to be more a problem of my competence in English than one of a confusion in the field, but I think there is a valuable point to be taken. The problem is that desire and intention are two different sorts of attitudes, used in different ways, and treatments of reasoning and problem solving which confuse the two lose much expressive power, power which is required both in deciding what actions to take next and in revising the program's mental state when actions are taken.

Desires and intentions are different in logical form. Desires aim at the satisfaction of some condition, and will be satisfied no matter how those conditions are brought about. Their content can be

[51] I believe that there are close connections between this view of plans and Minsky's K-line theory of memory [Minsky 1979]. For him, K-lines are ways of reactivating partial mental states. These connections are recursively arranged, in that activation of one K-node typically leads to the activation of several component partial states of mind. While we will use plans by making separate instantiations of them each time they are used, the analogy with K-lines becomes strong if we assume that plans are only stored once in memory, and "lit up" whenever they are needed rather than making multiple instantiations. If this is the case, then the definitional connections between plans and the subplans they define becomes very similar to that between K-nodes and the sub-nodes they activate.

stated roughly as "Condition X obtains." Intentions, on the other hand, must be satisfied in a certain characteristic way. Just what is the exact nature of intentions and their characteristic way of satisfaction has been the subject of much study. Harman [1976] and Searle [1979], for example, analyze intentions as self-referential attitudes, whose content is roughly "I take some action to attain condition X by way of carrying out this very intention." Intentions can be satisfied (at least partially) by trying, by taking actions on the basis of the intentions, whether or not the attempts succeed in attaining their aim or not. If the action fails, one forms another intention of the same sort. Attempts, however, have no bearing on the satisfaction of desires. In this sense it is much more difficult to tell if a desire has been satisfied than an intention, for the former requires verifying the effects of an action, where the latter requires only the proper mode of taking the action.

Desires and intentions also differ in other qualities ascribed to them. Different desires may have different relative strengths, which reflect the order in which, other things being equal, intentions will be formed to pursue the desires. For intentions, however, it makes no sense to speak of relative strengths. Once formed, and intention is an intention. There is no magnitude involved. Instead, two intentions may be related by other intentions about their relative priority of achievement, intentions to the effect that one intention should be carried out prior to another one. However, both desires and intentions share (along with beliefs) relative strengths of tenacity with which the program resists their abandonment. One might have, for example, two desires, the first of which is stronger than the second, but the second of which is held more strongly than the first. In this case, while the program considers the second desire less pressing it would rather give up the first desire than the second. Similar considerations apply to beliefs and intentions.

It is very important to distinguish between intentions and desires. For example, when modifying its plans, the program must analyze the causes and worth of the effects of the plan in action. An effect of an action can be either (1) both desired and intended (the normal case), (2) intended but not desired (action taken by compulsion), (3) desired but unintended (a serendipitous effect), or (4) unintended and undesired (an error or unwanted side-effect). In these four cases we assumed that undesired implied the opposite desire, but that is not correct, so there is actually a larger, more refined set of cases. But the important point is that how the program should modify the procedure depends on what classifications it makes of the procedure's effects. Serendipity might be used to construct new procedures specifically for realizing the desired effects, while errors normally call for patching the procedure to avoid the effects.

Thus the notions of desire and intention capture separate, useful ideas about rational thought and action, and the following part of this chapter and the next chapter will make that even clearer. Rather than confuse matters by using the ambiguous term goal, we abandon it for these more useful notions.

Unfortunately, none of the plans given in this thesis will seem to motivate this distinction terribly much. Most of the plans will be rather deliberate constructions which proceed step by step by means of intentions. One example of a plan employing desires is a problem-solving plan similar to problem-reduction problem solving. Given an intention to solve some problem, this plan would look for beliefs which say something about the problem statement, for example, $A \wedge B \supset PS$. The plan would then add desires for A and B, so that there would be lots of desires around for possible partial solutions. This would place the burden of controlling this solution effort in deciding which desire to pursue. Alternatively, desires might have been avoided by shifting the burden to a decision of which implication to use in the problem reduction, and then creating intentions rather than desires. The former method might be preferable if the problem is so difficult that the program must use information discovered in pursuing one desire in satisfying desires stemming from different reductions. The intention-based strategy does not make this opportunistic behavior as easy, since the several alternative solution paths are not being kept in mind simultaneously. Keeping all potential solutions in mind corresponds to using desires, while using intentions in this case corresponds to single-path explorations.

## 4.4 Desires and Intentions

The representations of desires and intentions consist primarily of three sorts of information: an *aim*, which is the condition to be achieved, *variable-mappings*, which identify in keyword fashion plan variables with variables and terms used in defining the aim, and *status* information. Desires and intentions are represented as theories. Each has as typed parts a set of input variables, a set of output variables, and an aim.[52] In addition, each also contains a statement of the form `DESIRE(theory)` and `INTENTION(theory)` about whether it represents a desire or an intention, redundantly repeating a similar statement in the global theory.

The aim is a theory describing the state of affairs desired or intended to be attained. We mean this to be a quite general notion, including, for example, descriptions of the program's own mental or physical state, and descriptions of changes in the world, that is, actions. Informally stated aims might be

(A) that the program believes the Banach-Tarski theorem,

(B) that the program has a proof of the Banach-Tarski theorem in ZFC,

(C) that the program rest its "arms" after moving the block halfway across the table,

(D) that the program buys some electronic parts,

(E) that the program finds out some information from someone,

(F) that the program is skilled at playing bridge,

(G) that the program leases a new tape drive from someone,

(H) that the program earns enough money writing novels to pay for its lease and to keep its programmer happy.

This research has not pursued the crucial problem of finding a language and vocabulary adequate for encoding all known information about the world, nor the encoding itself. We instead rely

[52] Variables are used for communicating information between activities. There may be better ways of doing this, but that is a subject for future study.

on work of others to build stores of information about the world and changes in it for use in aims of desires and intentions.

For example, suppose we have an intention to find the difference between two numbers X and Y.

```
INT-1: (the intention)
        Axiom: INTENTION(T-1);
        Typed-part INPUTS SET;
        Typed-part OUTPUTS SET;
        Typed-part AIM ADDER;
INT-2: (INPUTS of T-1)
        Typed-part X VARIABLE; (T-5)
        Typed-part Y VARIABLE; (T-6)
INT-3: (OUTPUTS of T-1)
        Typed-part Z VARIABLE; (T-7)
INT-4: (AIM of T-1) (from ADDER)
        Individual-constant A1;
        Individual-constant A2;
        Individual-constant SUM;
        Axiom: A1 + A2 = SUM;
INT-5: (from VARiABLE)
        Individual-constant VALUE;
INT-6: (from VARIABLE)
        Individual-constant VALUE;
INT-7: (from VARIABLE)
        Individual-constant VALUE;
INT-1: (again)
        [VALUE X INPUTS] = [SUM AIM];
        [VALUE Y INPUTS] = [A1 AIM];
        [VALUE Z OUTPUTS] = [A2 AIM];
```

Here sets are represented as theories where the elements are used as names of constants. This allows the same name to be used both for an input and an output variable, as it can be distinguished by the set it is in. Variables are also represented as theories, in which values are represented as attachments to the symbol VALUE. Theories representing variables will be used to note other information as well, such as whether there is a value or not, hence this complicated representation. The reason for using an explicit keyword mapping system in which equality axioms are used to identify intention variables with aim variables is so that the same aim may be used in several sorts of intentions, according to different input-output specifications of variables. For example, the ADDER aim used above can be used to specify several sorts of intentions including subtractions (X-Y=Z; X=SUM, Y=A1, Z=A2), addition (X+Y=Z; X=A1, Y=A2, Z=SUM), and doubling (X+X=Y; X=A1, X=A2, Y=SUM).

In addition to the flexibility of aim use allowed by the keyword variable mappings, desires and intentions can also include local modifications to their aims. Since their aims are just theories, axioms can be added to the aim theory. For example, a doubling intention might be specified either as

```
INT-1:
        Axiom: INTENTION(T-1);
        Typed-part ADDER ADDER;
        X = [A1 ADDER];
        X = [A2 ADDER];
        Y = [SUM ADDER];
```

or as

```
INT-1:
        Axiom: INTENTION(T-1);
        Typed-part ADDER ADDER; (T-2)
        X = [A1 ADDER];
        Y = [SUM ADDER];
InT-2: (ADDER)
        Axiom TIED: A1 = A2;
```

In the first case, the value to be doubled is given to the adder twice. In the second case, it is transferred only once, but the copy of the adder theory is modified to be a doubler.

Desires and intentions also contain information about the state of the process of their execution, for example, whether the desire or intention is being worked on, is yet to be worked on, or has been finished with. Here we distinguish between desire and intention in interpreting just what these status indicators mean. "Being worked on" means roughly "is being pursued with an intention" for desires, and "is being carried out by a primitive or a plan" for intentions. Since intentions are carried out by complex sequences of program operations, the most precise description of the state of the executing program is just the current step and environment of the code of the interpreter or whatever program is carrying out the intention. However, such a description is hopelessly detailed for normal use. In fact, the program employs several sorts of interpreter, and each of them would give a different report of the state of the execution process. Rather than use such an overly detailed indicator, eight major classifications of desires and intentions are used which summarize some of the most important aspects of their execution. Better classifications undoubtedly await discovery, but this initial list will suffice in this thesis. The classifications are as follows. (Figure 8 summarizes the transitions each intention goes through.)

1. *Progress status:* Initially, desires and intentions are *pending*. When the interpreter is working on one, it is *active*. When the interpreter is done with it, it is *finished*. The program is working on an intention if it is executing some primitive to carry out the intention, or has added a plan to the current state of mind to carry out the intention. The program is working on a desire if it has formed an intention to realize the aim of the desire. Desires are finished when their aim has been achieved, and intentions when some plan or primitive is completely executed to carry out the intention.

2. *Missing input values status:* A desire or intention either has values known for all of its input variables, or it is missing some input variables values. The interpreter will not begin work on ones missing some of their input values.

3. *Uncompleted predecessors status:* Desires and intentions are related in two partial orderings, desire strengths and intention priorities. Work on one cannot begin until all of its predecessors have been completed (specifically, are in the enabling-successors status described below).

4. *Uncompleted superiors status:* Desires and intentions are related in teleological relationships in which subordinate desires and intentions are used to carry out superior intentions. Work cannot begin on the subordinates until all superiors of the subordinates have been completed (specifically, are in the enabling-subordinates status described below).

5. *Enablement status:* Desires and intentions are *blocked* if they have missing input values, uncompleted predecessors, or uncompleted superiors. Otherwise they are *enabled*. The program will not begin work on blocked desires or intentions.

6. *Realization status:* An intention is said to be *realized* if it has been carried out by executing a

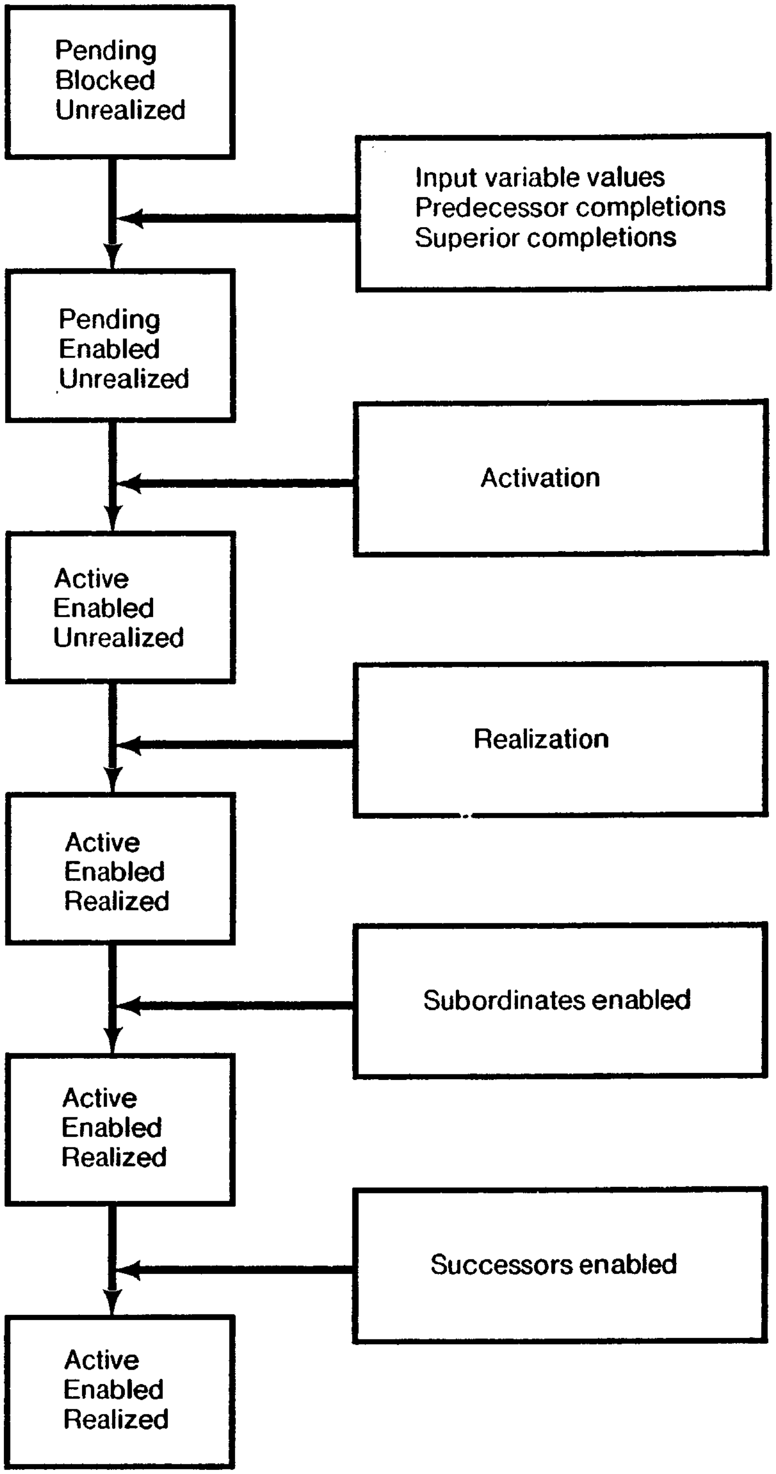


Figure 8
Progress Status Transistions

primitive program or by reducing it to a plan. A desire is realized if an intention to pursue its aim has been formed. They are *unrealized* otherwise.

7. *Enabling subordinates status:* Once an intention is active, normally after it has been realized by reduction to a plan, the interpreter will enable its subordinates if possible. This status indicates whether the intention should still block its subordinates or not.

8. *Enabling-successors status:* After a desire or intention has been realized, the interpreter may try to enable its successors if possible. This status indicates whether it should block its successors or not. The interpreter will declare an intention to be enabling-successors either if it was carried out by executing a primitive, or if its *main* subordinate (see below) has finished.

We represent all these sorts of status information in the desire or intention theory itself.[53] Each status name is a symbol in the language, and the possible conditions of the status are represented as possible attachments. Relationships between the possible attachments are represented as justifications.

In detail, each theory contains individual constants as in the following example intention.

```
INT-1:
        Axiom: INTENTION(T-1);
        Individual-constant MISSING-INPUT-VALUES-STATUS;
        Individual-constant UNCOMPLETED-PREDECESSORS-STATUS;
        Individual-constant UNCOMPLETED-SUPERIORS-STATUS;
        Individual-constant ENABLEMENT-STATUS;
        Individual-constant PROGRESS-STATUS;
        Individual-constant REALIZATION-STATUS;
        Individual-constant ENABLING-SUBORDINATES-STATUS;
        Individual-constant ENABLING-SUCCESSORS-STATUS;
```

The possible attachments and their standard justifications are as follows. (They are simplified somewhat for clarity.) The standard justifications are arranged so as to default the attachments to the appropriate values in the correct temporal sequence.

[53] Properly, perhaps, this information should be viewed as annotation on the theory in some more general theory (such as the plan containing the theory, or the global theory), but for simplicity of the representation we include it in the desire or intention theory itself.

```
N-1 Attach MISSING-INPUT-VALUES-STATUS SOME (justified as specified below)
N-2 Attach MISSING-INPUT-VALUES-STATUS NONE (SL () (N-1))

N-3 Attach UNCOMPLETED-PREDECESSORS-STATUS SOME (justified as specified below)
N-4 Attach UNCOMPLETED-PREDECESSORS-STATUS NONE (SL () (N-3))

N-5 Attach UNCOMPLETED-SUPERIORS-STATUS SOME (justified as specified below)
N-6 Attach UNCOMPLETED-SUPERIORS-STATUS NONE (SL () (N-5))

N-7 Attach ENABLEMENT-STATUS BLOCKED (SL (N-1) ()),
        (SL (N-3) ()), (SL (N-5) ())
N-8 Attach ENABLEMENT-STATUS ENABLED (SL () (N-8))

N-9 Attach PROGRESS-STATUS PENDING (SL () (N-10 N-11))
N-10 Attach PROGRESS-STATUS ACTIVE when activated: (SL (proc) (N-11))
N-11 Attach PROGRESS-STATUS FINISHED when finished: (SL (proc) ())

N-12 Attach REALIZATION-STATUS REALIZED when realized: (SL (proc realization) ())
N-13 Attach REALIZATION-STATUS UNREALIZED (SL () (N-12))

N-14 Attach ENABLING-SUBORDINATES-STATUS YES when so: (SL (proc) ())
N-15 Attach ENABLING-SUBORDINATES-STATUS NO (SL () (N-14))

N-16 Attach ENABLING-SUCCESSORS-STATUS YES when so: (SL (proc) ())
N-17 Attach ENABLING-SUCCESSORS STATUS NO (SL () (N-17))
```

In the above justifications, proc stands for the procedure adding the justification. Realization stands for the record of the realization of the desire or intention, that is, either the plan or action record that the interpreter constructs (as explained in Section 4.10) for intentions, or the intention constructs from a desire.

Justifications for N-1, N-3, and N-5 above involve statements in other theories. Recall that each variable is represented as a theory and the value as an attachment in that theory to the symbol VALUE. In addition, we have each variable theory contain a constant VARIABLE-HAS-VALUE. Whenever an attachment is made to VALUE, thus specifying a value, we by convention also use that attachment to justify an attachment of YES to VARIABLE-HAS-VALUE. Symbolically, we typically have justifications as follows.

```
N-18 Attach VALUE xxx (some justification)
N-19 Attach VARIABLE-HAS-VALUE NO (SL () (N-20))
N-20 Attach VARIABLE-HAS-VALUE YES (SL (N-18) ())
N-1 Attach MISSING-INPUT-VALUES SOME (SL (N-19) ())
```

This last justification, when made for each input variable, ensures that the MISSING-INPUT-VALUES-STATUS will be properly maintained. Similarly, N-3 and N-5 above will have justifications involving other desires and intentions. N-3 will be justified in terms of an ordering relationship and the enabling-successors-status attachments of the predecessor. N-5 will be justified in terms of a subordinate relationship and the enabling-subordinates-status attachments of the superior.

Finally, desires and intentions contain *scope* information about the context of their definition. The parent theory of each is either the plan it is defined in or the current state of mind. The desire or intention theory, in addition, is justified in terms of the parent and the procedures adding it to the current state of mind.

## 4.5 Policies

The intentions presented in the previous section all had aims describing some action that the program could decide to carry out. However, not all intentions can be expressed in that form. Instead, there are intentions with conditional or hypothetical statements as their aims. For example, the program can decide to carry out "I intend to visit George," but not "I intend to visit George whenever I am in New York." This latter intention we term a *policy.*[54] In the following, all policies will be intentions. There may be desires with hypothetical statements as their aims, but I have not yet worked out how they might be used, and so leave them an open problem.

Policies are represented as theories similar to other intentions. Policies have sets of input and output variables, an aim, status information, and a scope or context of definition just like other intentions. In addition, policies are distinguished by the program from other intentions by a statement POLICY(policy) in them, where policy is a symbol referring to the policy theory itself.

The aims of policies are instances of a "conditional" theory, the prototype of which is

[54] [McDermott 1978] introduced this technical meaning of policy as an intention with a hypothetical aim.

```
IN CONDITIONAL:
        Individual-constant CONDITION;
        Individual-constant ACTION;
```

The aim of a policy is a copy of this theory in which CONDITION is attached to a sentence wff and ACTION is attached to the theory describing the action as in the aims of ordinary theories.[55]

Where ordinary intentions usually are only active for some limited duration, and then are carried out, policies need not be so limited. Some policies will be of limited scope, for example, while the plan they are part of is being executed, or while some intention is active. But other policies may have unlimited scope, that is, some might be constantly in effect until a decision is made to abandon them.

As we will interpret them, policies embody intentions to make decisions in certain ways. Where intentions ordinarily are intentions to *act* in certain ways, policies embody intentions to *reason* in certain ways. Instead of leading to actions, policies lead to reasons for possible actions in decision-making. Thus we would translate the informal intention "I intend to visit George whenever I am in New York" as the intention to reason that I ought to visit George if I am in New York deciding what to do, that is, the intention to construct the option of visiting George and a reason for taking that option as the outcome of the decision.

This interpretation of policies has two major consequences. The first consequence is that it allows some flexibility in carrying out intentions. If I have intentions to visit George and to buy books whenever I am in New York, I do not feel compelled to do either the minute I arrive there. Instead, these intentions merely suggest the possibilities of visiting George and of buying books, and construct reasons for taking those actions. But since these are just reasons for action rather than absolute requirements, I can defeat these reasons in this decision and do something else, and reconsider the possibilities the next time I think of what to do. Since specific cases of their actions can be defeated in this way, policies seem

[55] I find this representation for policy aims unsatisfactory, but have not yet found how to improve it.

similar to what have been termed *prima facie* obligations in the literature.[56]

Second, this interpretation of policies means that they embody some of the values of the program. That is, we would translate a preference of one possible action over another in some circumstances as the intention to reason for the first and against the second in such circumstances, specifically, to defeat reasons for the second possibility with reasons for the preferred possibility.

What are policies for? In the following we will use them in many ways. Policies will express temporal ordering relationships between intentions, as in the intention to carry out one intention before another, which we can interpret as the intention to choose the prior intention over its successor when deciding what to do if the prior intention is yet unrealized. Policies will embody the strengths of desires, where we interpret one desire as stronger than another when the option of working on or satisfying the first is preferred over working on or satisfying the second in decisions of what to do next. Policies will embody many of the preferences of the program, such as those used in belief revision to choose one possible revision over another. There policies amount to statements of the strength or commitment to beliefs.

With this interpretation of policies, we see the special importance of the scopes of policies. Policies of temporary duration amount to temporarily adopted values. Policies of unlimited duration amount to permanent values. In this way, RMS serves the function of maintaining the current set of values as well as the current sets of other attitudes. And, as Chapter 6 discusses, permanent values can be adopted or abandoned through decisions to create or defeat policies of unlimited scope.

Policies, like other intentions, are carried out either by executing primitive programs or by reducing them to plans. The next chapter discusses something of how and when policies are carried out during deliberations, but the details of this, and the details of how the progress statuses of policies are manipulated, are yet to be worked out.

[56] The term is due to Ross [1930]. See also [Harman 1977] and [Searle 1978].

### 4.6 Relationships Between Desires and Intentions

In addition to the plan steps embodied in ordinary desires and intentions, plans also contain policies which restrict how the steps are to be carried out. For example, the program might have not only the intention (1) to place block A on top of block B, and the intention (2) to place block B on top of block C, but also the intention (3) to carry out the previous intentions in the order (2 then 1). Another example would be the intention (4) to build a tower of blocks, and the intention (5) to use intentions (1,2,3) as a way of carrying out (4). As these examples suggest, the two main sorts of inter-step relationships are ones which impose (relative strength or temporal) orderings on the realization of desires and intentions, and ones which describe teleological relationships between desires and intentions.

Actually, relationships between intentions are always teleological. Teleological relationships, preeminently those of one intention being a prerequisite of another or of one intention being a way of carrying out another, figure crucially in all other relationships. For example, two intentions might be executable in either order. If one order is more efficient than another, then that is a reason making for a temporal ordering on them, but the underlying explanation remains the teleological one of the efficiency of the computation. Similarly, if the second intention depends on some precondition being achieved by the first intention, then one would again have a temporal ordering policy, with the underlying reason being the teleological relationship of prerequisite.

In spite of the fundamentally teleological nature of rational intention relationships, we separate out the ordering relationship so that they may be specified even when (as is usual in informal program efficiency arguments) the reasons behind the relationship still have not been completely formulated. In addition, this separation permits us to use uniformly an ordering relationship on both desires and intentions. For intentions the order is temporal order, and for desires the order is relative strength. Ordering policies never connect both desires and intentions, as these are different sorts of entities, between which an order makes no sense.

All policies of these sorts are defined as copies of one of the standard policy types with the desires or intentions involved added into the theory as attachments. For example, one of the main temporal ordering policy types is that of one intention anteceding another. This is defined by the following theory.

```
IN ANTECEDENCE-POLICY-THEORY:
        Axiom: POLICY(ANTECEDENCE-POLICY-THEORY);
        Typed-part INPUTS SET;
        Typed-part OUTPUTS SET;
        Typed-part AIM CONDITION-THEORY;
        Individual-constant ANTECEDENT;
        Individual-constant SUCCESSOR;
        Axiom: ANTECEDES(ANTECEDENT, SUCCESSOR);
IN T-1: (Aim of ANTECEDENCE-POLICY-THEORY)
        Attach CONDITION ¬SUCCESSORS-ENABLED(ANTECEDENT);
        Attach ACTION (CON (OPTION SUCCESSOR) (SL (ANTECEDENCE-POLICY-THEORY) ()));
```

A policy of this sort could then be created relating Intention-1 and Intention-2 by making a copy of ANTECEDENCE-POLICY-THEORY and adding two attachments in its aim, that of ANTECEDENT to Intention-1, and SUCCESSOR to Intention-2.

To make the interpreter more efficient, we also include in the desire or intention theory lists of all ordering policies mentioning it. For example, each has the individual constants ANTECEDENT-POLICIES and SUCCESSOR-POLICIES, to which are attached lists of all antecedence policies mentioning the desire or intention as the successor or antecedent, respectively. The policies themselves are kept in these lists rather than just the antecedents or successors so that the policies may be used in justifications. Also, whenever new ordering policies are added, corresponding justifications for the status attachment of the desires or intentions are added, to facilitate reasoning about which successors are blocked by the order relationship. This duplicates some of the reasoning that would normally occur in deliberations in a convenient and efficient, but still defeasible, fashion.

The major types of policies relating desires and intentions are order, dataflow, prerequisite, and subordinate policies.

1. *Ordering policies:* As mentioned above, ordering policies represent intentions to realize desires or intentions in certain ways, to relate the steps of processing each of those desires or intentions. Of course, the descriptions of the steps of carrying out an intention might be very detailed, so these policies might specify very complex relationships. For example, specifying the temporal interleaving of coroutines, or tasks like laying and finishing a concrete driveway, can be very complicated, because one does a little of one, a little of the other, more of the first, and so on until they are finished. We avoid such complexity in this thesis, and leave the problem of developing a more complete vocabulary for execution relationships for future research. Instead, we present merely a small set of concepts for relating two desires or intentions.[57]

If I1 and I2 are two intentions, we denote the times at which the processes carrying out these intentions begin and end, abbreviated B1, E1, B2, and E2. We can identify the beginning of a intention as the time of transition of its progress status from pending to active, and the ending of a intention as the time of the following transition from active to finished. With these terms, we define the temporal ordering policy types as follows.

| | | |
|---|---|---|
| I1 precedes I2 | Directly I1 finishes, I2 begins | B1<E1=B2<E2 |
| I1 antecedes I2 | I1 finishes before I2 begins | B1<E1<B2<E2 |
| I1 leads I2 | I1 begins before I2 begins | B1<B2 |
| I1 overlaps I2 | I2 begins during I1 | B1<B2<E1 |
| I1 covers I2 | I2 occurs during I1 | B1<B2<E2<E1 |
| I1 beats I2 | I1 finishes during I2 | B2<E1<E2 |

If D1 and D2 are desires, we say that D1 antecedes D2 to mean that D1 is a stronger desire than D2 in the partial strength order.

If circularities are present in the ordering policies, so that an inconsistent set of orders exist, then

[57] Many people have studied and are studying this question of vocabularies for execution relationships. See the literatures on parallel programming languages [Hewitt 1977], petri nets, and PERT extensions [Wiest and Levy 1977]. Smith and Davis [1978] and Kornfeld [1979] study such vocabularies in terms of parallel problem-solving systems.

One significant extension to our vocabulary might involve the introduction of a clock or time-system for referring to future events not related to specific actions. This sort of extension would be necessary for stating intentions like "I intend to finish this thesis by May 12, 1980."

some desires or intentions will all be blocked in a deadlock. To avoid this, when the program reflects on its current plans to decide what to do next, it also checks to see if such a deadlock exists. (Actually, whenever an ordering policy becomes one of the current policies, the program checks to see if it is consistent with the previous ordering.) If an inconsistent ordering is detected, the program sets itself the intention of breaking the deadlock by abandoning one or more policies. It makes the decision of which policies to abandon by using the deliberation techniques described in the next chapter and the guidelines described in Chapter 6.

2. *Dataflow policies* represent intentions to use the outputs of one desire or intention as the inputs to another, that is, the intentions to infer values for some variables upon getting values for other variables. In their representation as theories, these policies mention not only the desire or intentions being connected, but also the input and output variables of each of each that are to be identified. Dataflow policies are respected by the interpreter by waiting until a value is computed for each input variable mentioned in a dataflow policy. Dataflow policies thus ensure that the producer leads the consumer by enough time to compute the required value. Dataflow policies are actually always carried out by a built-in primitive which propagates these values when necessary. To make this easier, each variable theory contains a symbol EQ-POLICIES which is attached to all dataflow policies mentioning the variable.

It might be useful to have other classes of dataflow policies, such as an analogue of "precedes" above, wherein one intention would begin immediately upon the availability of some variable value. This might be the case with removal of intermediate stage waste in a complex chemical process. However, it would seem difficult to implement this sort of policy without some form of actual parallelism in one's machine, since the producing intention may produce the value while in an uninterruptible stage of its process.

3. *Prerequisite policies* make explicit the rationale of temporal orderings. Prerequisite policies mention at least two intentions, rather than only two. They are interpreted as the intention to use the several effects of one set of intentions I1, ..., Ik as the means of achieving a combined state of affairs prior

to another intention I. Along with each of the intentions the prerequisite policy mentions a logical formula expressing the corresponding state of affairs. According to the interpretation, if we write the formula corresponding to an intention I as F(I), the meaning of the policy is that

$$\mathtt{F(I1)} \wedge \ldots \wedge \mathtt{F(Ik)} \supset \mathtt{F(I)},$$

or perhaps

$$\mathtt{\langle I1\rangle F(I1)} \wedge \ldots \wedge \mathtt{\langle Ik\rangle F(Ik)} \wedge (\mathtt{F(I1)} \wedge \ldots \wedge \mathtt{F(Ik)} \supset \mathtt{F(I)}) \wedge (\neg \mathtt{F(I)} \supset \mathtt{[I]false}).$$

Here we have written a formula in dynamic logic [Harel 1979], in which $\mathtt{\langle action\rangle P}$ means that $\mathtt{action}$ can achieve a state in which P holds, and $\mathtt{[action]false}$ means that $\mathtt{action}$ cannot terminate in the current state.[58]

4. *Subordinate and reduction policies* make explicit intentions to use one set of desires and intentions as a means of carrying out another. Whenever the interpreter reduces an intention to a plan, it adds an instance of the plan to the current state of mind and adds a reduction policy intention the intention and the plan. It also adds subordinate policies relating the reduced intention and each of these new desires and intentions. In contrast to prerequisite policies, which state that the *preconditions* of an intention are attained jointly by its predecessors, the reduction and subordinate policies state that the *effect* of the reduced intention is attained jointly by its subordinates.

5. *Main subordinates:* A subordinate intention of an intention may be annotated as the main subordinate of the intention. This policy represents the intention to complete the main subordinate before beginning work on the intention's successors. For example, if the plan for serving dinner has two steps, to prepare the food and then to serve it, the preparation step involves the substeps of cooking the food and then washing the pots and pans. But the food may be served just after cooking the food, and the

[58] It probably is simply wishful thinking to apply a language as precise and as inexpressive as dynamic logic to discussing actions as general and as vaguely specified as plans, but some language is needed for this purpose. Dynamic logic is much too limited except as a basis, for we need to be able to discuss in the language itself algorithmic complexity, intermediate states, relations between actions, etc., none of which are fully within dynamic logic's realm. Moore [1979] explores a logic of action with the power to treat actions as objects, but he makes no use of that power, and restricts his study to actions as in dynamic logic. Hayes [1971] explores a logic of actions which attempts to capture statements about the causal relations between objects affected by actions.

washing up can be postponed until after serving. (See Figure 9.) Main subordinate policies thus serve a function analogous to that of dataflow policies, but concerned with action effects rather than variable values.

Main subordinate policies are specified by method statements, as described later. They mean that all of the intentions in the plan must be carried out, but the superior itself will be carried out once the main subordinate has finished, that is, from the point of view of the superior, all remaining subordinates are merely cleanup steps unrelated to the purpose of the plan.

One extension of this idea would be to have multiple main steps of plans, each of which allows a different set of successors of the superior to proceed. However, this would require taking into account considerable information about the context of the superior. For simplicity, we restrict the program to single main subordinates, and leave the generalization for future studies.

### 4.7 The Hierarchical Structure of Plans

The preceding pages have explained two major classes of constituents of plans, namely desires and intentions. Some of these specify the steps of actions, and others restrict how the former are to be realized. But plans contain many other sorts of information whose purpose is to fill out, refine, and make coherent the behavior sketched out by the desires and intentions. In addition, the plan itself is an object in a library of plans, and plans contain information aiding in their indexing in this library. Plans are represented as theories with a number of standard parts. Plans have a set of input variables, a set of output variables, a set of desires, a set of intentions, a set of subplans, a set of assumptions, and a set of plan definitions to be held during the tenure of the plan. Concretely, a plan theory will have the following parts as well as further restricting axioms.

```
Typed-part INPUTS SET
Typed-part OUTPUTS SET
Typed-part DESIRES SET
```

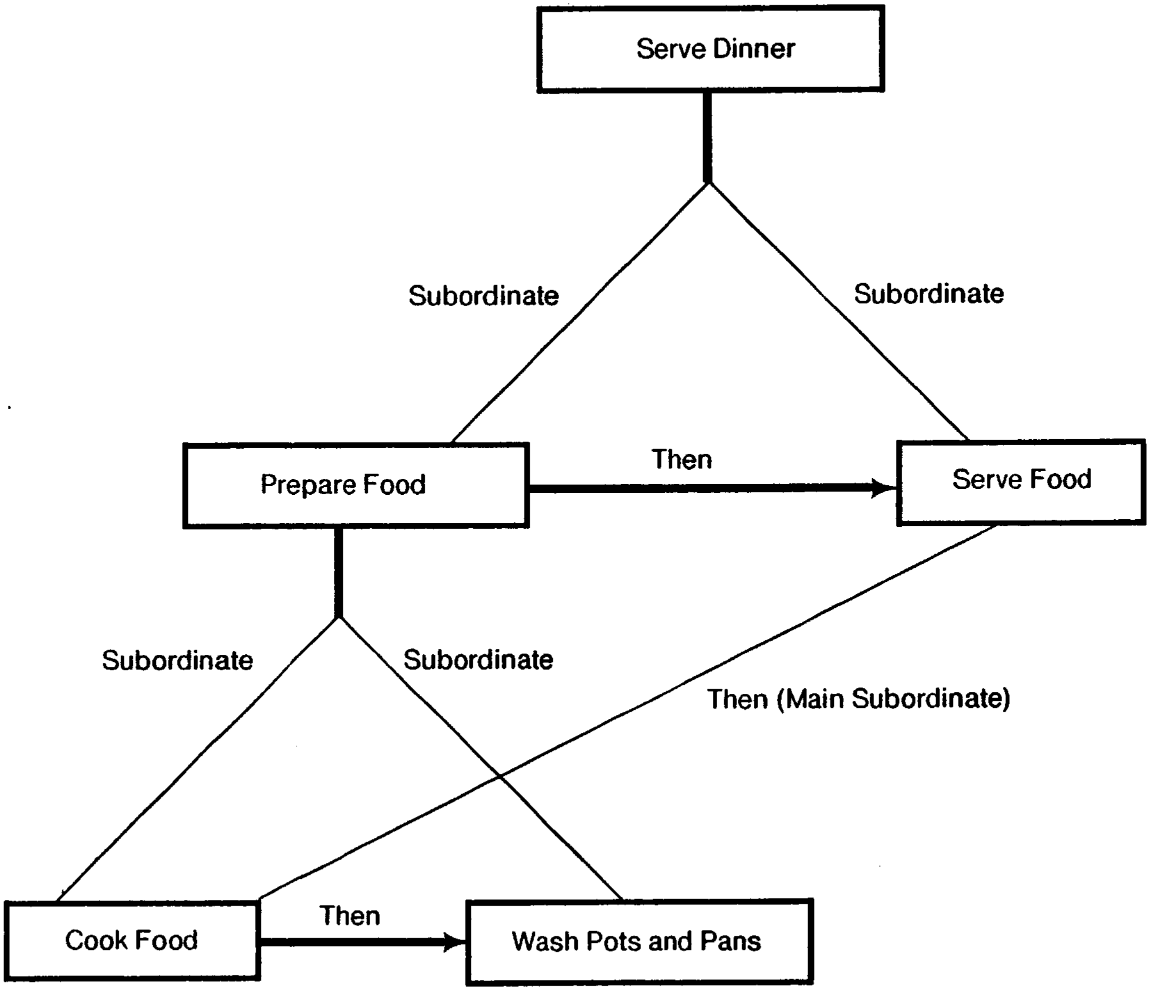


Figure 9
Plan for serving dinner

```
Typed-part INTENTIONS SET
Typed-part SUBPLANS SET
Typed-part ASSUMPTIONS SET
Typed-part PLAN-DEFINITIONS SET
```

All of the subparts of a plan have names. The set of desires of a plan has names for each of the desires, with the desire theories attached to these names. Similarly, each subplan in the set of subplans, each assumption in the set of assumptions, each intention in the set of intentions, and each plan definition in the set of plan definitions may have names. The input and output variables have names of course, and the program generates names for any assumptions, policies, and plan definitions entered anonymously by the syntactic macros described in Section 4.12. The naming of these parts allows, for example the combination of copies of two plans from the plan library for incorporation into the current plan, or the defeat of a local assumption specifically by a local policy.

As in desires and intentions, the variables of a plan are theories, with the same conventions. Since the plan is used as a unit of behavior by the interpreter and by other plans, it is crucial that the details of the plan's construction normally be hidden. This is the function of the plan variables. The plan's input and output variables will be the only parts of the plan normally referred to by other plans. These variables will be connected to the variables of the desires and intentions by dataflow policies. For example, whenever a plan is built from a subplan, it is necessary to provide variable mapping information in dataflow policies to connect the relevant plan variables with the relevant subplan variables.

Plans often contain restricting axioms which modify the subplans used in constructing a plan. For example, one frequent modification is attaching constant values to variables of subplans.

Plan theories may contain a number of assumptions. These are beliefs to be held during the execution of the plan to be retracted if contradictions are encountered. For example, when negotiating to buy a house, one typically assumes that the seller will sell the house once agreeable terms are reached. Another sort of example is the specification of default values for local variables or other variables, an instance being a plan to clear the top of a block which assumes that the table is always a good default target location for any blocks to be moved. A final sort of assumption is that of assumed method relationships between procedures and aims (as explained below), in which it is assumed that some procedure is relevant to achieving some aim during the plan's execution.

Plan theories may contain a number of policies to be in force during the plan's execution to influence the expected sorts of decisions. These are typically concerned with decisions about the order in which the plan's desires and intentions should be carried out, the methods by which they should be carried out, and the ways that the plan's assumptions should be revised in case of difficulties. For example, one's plan for giving a talk may include the policy to prefer to answer questions with "I don't know" rather than trying to think on one's feet. Similarly, the cooking with grease plan mentioned earlier might employ a policy to change the default plan for extinguishing fires to one involving a fire extinguisher.

Finally, plan theories may contain a number of plan definitions to be held during the plan's execution.[59] An example is the plan for cooking with grease mentioned earlier, which contains a local plan definition for how to put out fires, along with a policy preferring the local plan to the standard plan. Locally defined plans and policies are how one might write plans with conditional steps. Each of the cases is encoded as a policy which adds the appropriate intention or plan to the network depending on what conditions hold.

Temporary assumptions of beliefs, policies, and plan definitions are actually shorthand for the intentions to adopt them temporarily. As intentions, they can be related by temporal ordering policies. For example, in Section 4.12, we present a plan which midway through its execution makes an assumption to endure only while carrying out the next intention. We separate out explicit sets of these assumptions as abbreviations both for the intention declarations and for the ordering policies necessary to make all the plan-extant assumptions precede all the "real" intentions. Temporary assumptions are made

[59] Actually, the variables and plan definitions are just temporarily defined concepts. The plan might contain other sorts of temporarily defined concepts, but variables and plan definitions are the most important sorts, so we concentrate on them.

to have scopes limited to the duration of the plan by making the assumed attitude depend on the statement that the scope intention is not finished. That is, the assumptions depend monotonically on the statement that the superior intention has become active, and non-monotonically on the statement that the superior has finished.

### 4.8 Plan Specifications

Plans are involved in at least three hierarchical organizations. The first of these is the hierarchy of construction, in which existing plans can be combined to construct a new plan. Second is the hierarchy of definition, in which plans can contain local definitions of other plans to be of limited temporal duration. Third is the hierarchy of effects or situations of use, in which plans are indexed by the purposes for which they can be used. This indexing information is divided into two components: specifications of plan effects, and method statements to connect ends with relevant plans as means.

Plan effect specifications are simply statements about the properties of the plan. For example, Section 1.4.2 indicated how statements about what procedures call other procedures could be used in answering questions about the program's history of actions. For another example, statements estimating the complexity of procedures can be used in planning under time constraints. But the most studied sort of statement of procedure properties is that of Floyd-Hoare specifications: pairs of formulas P and Q with the interpretation that if P holds before the plan is executed, then Q will hold if the plan terminates, where termination of the plan is not assumed. These specifications take the form $\mathtt{P \supset [plan]Q}$ in dynamic logic [Harel 1979], and termination can be correspondingly expressed as $\mathtt{P \supset \langle plan \rangle true}$. There can be several plan effect specifications for each plan. These specifications are not used in the normal operation of the program, but are useful in hypothetical reasoning and in modifying or analyzing the plan library. In hypothetical reasoning the technique of symbolic execution is used. This technique does not execute a primitive or plan, but instead tries to prove that the antecedents of a procedure's specifications hold in one situation, and if successful, then concludes the consequents in the following temporal situation. In modifying the plan library, the program might seek, for example, to reorganize the plans to make sure that they are seen to be relevant to problems whose statements are contained in the plan's effects. In analyzing the plan library, the aim is to more completely annotate (and verify the correctness of) the plans and their internal structure with the records of, say, additional prerequisite policies where before there were only temporal ordering policies.[60]

Information involving a plan's effects is more directly useful in the form of *method statements.* These indicate what plans are useful for which aims of intentions. The interpreter uses method statements to retrieve the plans and primitives relevant to achieving the aim of the intention being interpreted. In addition, method statements for plans also specify which step of the plan is the main step with respect to the desired effect. Thus a multistep plan may be a means of achieving several sorts of aims. Each of these uses of the plan would be specified in a separate method statement, along with a statement of which of the steps of the plan achieved the partiLular effect (aim) of relevance.

Method statements are represented as simple beliefs of the program. For plans they take the form

```
PLAN-METHOD(aim, plan, mainstepname),
```

where `aim` and `plan` refer to, respectively, a theory describing some aim concept and a plan theory. `Mainstepname` is the name of some desire or intention in the plan which is declared main. For example, a method statement like the following might be used in describing the subplan of the dinner-serving plan of Section 4.6.

```
PLAN-METHOD(PREPARE-FOOD-AIM, COOK-THEN-WASH-PLAN, COOK-FOOD).
```

[60] [Shrobe 1979a] discusses such techniques in detail.

For primitives, method statements take the form

```
PRIMITIVE-METHOD(aim, primitive).
```

Methods relevant to an intention's aim are retrieved by procedures which take the aim, instantiate it with the intention's input variable values, and then look in the procedure library for method statements which mention aim types subsuming the particular instantiated aim. This can be a very difficult problem, as many inferences might be required to judge one aim description subsumed by another. This is an incompleteness in the current program. I envision actually employing several sorts of retrieval procedures, simple ones which are fast but miss some methods (for example, ones which just look up the VC hierarchy from the particular aim) to procedures which are slower but find more of the relevant methods. Different versions of the interpreter would then use the different retrieval procedures, and in difficult cases, self-apply the program to retrieving the relevant methods.

This issue of what methods should be retrieved as relevant to a particular aim seems to be one to which deontic logic is relevant. One of the issues addressed by logics of commands and obligations is that of what commands and obligations are entailed by a given command or obligation. For example, suppose I am obliged to visit MIT. Since MIT is part of Massachusetts, being on the grounds of MIT entails being in Massachusetts. Thus we can infer that I am also obliged to visit Massachusetts as well. I suggest that this question of entailment of commands or obligations is closely connected with the question of what method aims entail or are entailed by a given intention aim. Further study of this connection might shed light on both the techniques of this thesis and on the proper role of deontic logic. The next chapter mentions another connection with deontic logic as well.

### 4.9 The Current State of Mind

The program represents its current state of mind to itself as the global theory ME. ME contains statements about the program's current concepts, reasons, beliefs, desires, and intentions. To act, the program reflects on the contents of ME, on what desires and intentions are currently held according to the global theory.

Plans reflect the structure of the program, as they are used to temporarily augment the current state of mind. Plans are concepts describing subconcepts (the plan variables and plan definitions), reasons, beliefs (the assumptions made by the plan), desires, and intentions. When the program carries out an intention by reducing it to a plan, it adds the contents of the plan to the current state of mind by making the global concept ME be a VC of the plan-instance concept. This VC statement (in ME of course) is justified monotonically in terms of the statement of the reduction, and non-monotonically in terms of the incompletion of the intention being reduced. In this way the contents of the plan augment the current state of mind until the execution of the plan (and hence its superior) has finished, or until the superior is abandoned. At that time, the VC statement becomes *out*, and the plan's contents are removed from the current state of mind.[61]

We leave several unanswered questions here. This technique for interpreting plans requires a distinction between the satisfaction of an intention and the finishing of an intention. Plans are

[61] It is often argued (e.g., by Taylor [1974]) that our notion of "self" is an illusion. Even if one acknowledges this thesis, the idea of one's self may be useful in practice, and in fact, people typically find the concept indispensible. However, people also voice their indecision with phrases like "Part of me wants to do this, part of me wants to do that," or "I'm of two minds about it." These highlight the next problem: Is there just one "self" of a person? Nagel [1979c] argues that there cannot be just one self from psychological evidence concerning brain bisections. Minsky and Papert [1978] argue against a single self both from psychological evidence concerning the development of intelligence in children, and from computational grounds, namely that presupposition of a single self begs the question of how the mind might work. They propose an analysis of the mind into many hundreds or thousands of simple "agents" in a "society of mind." The mind's idea of its self continually changes as different agents gain control. The proposals of this section for the current state of mind might be viewed as one realization of Minsky and Papert's ideas. In the program, each procedure actually carries with it a fragment of the current state of mind, so what the current state is varies with what procedures have control. Thus Minsky and Papert's [1973] conservation examples, in which the physical laws believed by the child seem to vary with the problem being worked on, can be explained easily by their suggestion of different beliefs embodied in the different procedures used by the child. Similarly, recognition of conflicts between two currently active procedures manifests as reflection to an arbitrating procedure which specifically considers which of the two "minds" (procedures) to adopt.

purposeless procedures, or more precisely, procedures which can be used for many different purposes. Because of this, plans may be indexed via method statements as useful for achieving intentions for which they are more general than necessary. One symptom of this is that of main subordinates, in which the aim of the intention is sometimes satisfied before all of the intentions in the plan have been carried out. In some cases this indicates that the remainder of the plan can be discarded, as when I use my plan for getting to someplace as a plan for getting to one of the stops on the way. But in other cases, the remaining intentions of the plan are clean-up steps which secure the results achieved by the main step, or which prevent certain undesirable side-effects. For example, my plan for checking if I turned off the lights in my dormitory room has a step for closing the door after I have opened it and looked inside. This step does not serve the nominal purpose of the plan, my intention to make sure the lights are out, but rather my policy of discouraging robberies by keeping my door closed. In this case, I cannot simply discard the remaining step of the plan after achieving its purpose.

We do not offer any way of overcoming this difficulty here. A suggestion for investigation is that the plan also contain a schematic reason for the last step in terms of the realization record of the first step (as explained in Section 4.10, this is a belief that the action was taken) and the extra-plan policy of keeping the door closed. However, just how this would work is uncertain, because presumably the reasons contained in a plan have tenure limited to that of the plan as well, so nothing has been gained. The plan might contain a step taking the action of adding the reason permanently. Alternatively, a distinction might be developed between the satisfaction of an intention and the finishing of an intention. Perhaps the plan's tenure and the finishing of the intention are coincident with the satisfaction (and simultaneous finishing) of the reduction intention to carry out the first intention by means of the plan.

## 4.10 The History of Actions

As the interpreter acts, it makes records of its actions so that later it can tell what it did and why. These records include beliefs about its past actions and the connections between these actions, the desires or intentions leading to them, and their effects (the changes in beliefs and other attitudes stemming from the actions). The records left by the interpreter include a *realization record* and a *realization statement*, where the realization record reflects what action was taken, and the realization statement reflects which intention the action realizes.

Realization statements are just beliefs in the global theory of the form

```
REALIZES(realization-record, intention),
```

and are justified by the interpreter procedure performing the realization and by the decision used to select the method for carrying out the intention. The intention and realization record contain redundant pointers to the realization statement to facilitate explanations.

Realization records are beliefs of the form

```
ACTION(plan/primitive, argument list)
```

where `plan/primitive` is the plan or primitive in the procedure library by which the intention was carried out, and the argument list is a list of the variable bindings used for plan variables or primitive arguments derived from the intention. Realization records are justified by the interpreter program alone. They are not conclusions drawn from other beliefs or attitudes, but rather are observations made by the interpreter about its own actions.

The nature of realization records can be clarified by comparing them with RMS justifications. Justifications are actually a form of realization records. The realization records specified above record actions for which explicit intentions exist. They record actions taken directly on the basis of intentions.

Justifications, on the other hand, are constructed by primitives called by other primitives. They record actions taken without explicit intentions, actions taken only indirectly on the basis of explicit intentions. Realization records and justification have similar forms. Recall that the justifications employed by the program are all reflected in explicit beliefs of the form (we only consider SL-justifications here)

```
SL-JUSTIFICATION(name, node, {name}+inlist, outlist)
```

The standard use of justifications includes the primitive's node in the *in*list and its arguments' nodes in either the *in*list or the *out*list, depending on how they are used in the procedure. Ignoring the name/node and *in*list/*out*list complexities, justifications share the form of realization records: procedure plus arguments. Justification record the unconscious inferential actions of the program.

It might well be possible to make the treatment of justifications and realization records both more uniform and more general, but that is left for future research.

Just as attitudes depend on the explicit belief about their justification, attitudes concluded from plan or primitive realizations depend on the realization records for those plans or primitives. Each new plan instance added to the current state of mind is justified monotonically in terms of a realization record. Each conclusion drawn from a primitive includes the realization record in the inlist of the justification for that conclusion. For example, if a primitive computes a value for one of the output variables of the intention it is carrying out, it justifies this attachment in terms of the realization record. If it computes a new value for some symbol (e.g the list of successors of an intention theory), it likewise justifies the new attachment in terms of the realization record, as well as using this record in a justification defeating the justification of the previous attachment. With such records, the program can discard the effects of an action if it discards the memory of the action, say by deciding that it had merely hallucinated the action. In more normal cases, the program can trace the causes of circumstances described by its beliefs by tracing backwards through the justifications of the beliefs, thus seeing part of their inferential sources, back to realization records, then through the realization statements and the justifications of the intentions, thus seeing part of their causal sources. We make use of this sort of analysis in Chapter 6.

The interpreter also makes statements of historical order relating the realization records. These are statements which tell the temporal order in which actions were taken. These statements are of the form

```
PRECEDING-ACTION(prior-realization-record, following-realization record).
```

Realization records also contain pointers to their preceding and following realization records in the temporal order. Such statements are redundant in some versions of the interpreter, as discussed below, when the interpreter records the order in which it acts on intentions. The meaning of these statements might be backed up by a theory of time. This would allow the program to reason about its history. For example, its theory of time might include facts about the transitivity of PRECEDING-ACTION, about the linearity of that ordering (if it is linear), about (as Section 1.4.2 suggested) the non-occurrence of deliberate actions which do not appear in realization records, etc. Just what the program's theory of time and its actions should be is still an open question. Rescher and Urquhart [1971] survey many temporal logics. Hayes [1970] (and to a lesser extent, also [McCarthy and Hayes 1969]) surveys temporal logics with an eye to applications in reasoning programs.

It is often possible to recover considerable information about the history of a particular attitude by examining the complete set of reasons concerning it. Since primitives change attitudes by defeating previous justifications on the basis of realization records, changes in the status of an attitude can be inferred from a justification for it in terms of one action, a justification defeating the first in terms of a later action, a justification defeating the second in terms of yet a later action, etc. It remains for future studies to pursue a careful development of such techniques.

However, some interpreters may not record temporal orderings of actions. Humans frequently cannot recall the order in which certain actions occurred, or that they took some action rather than another, or that they took some action at all. These failures need not all be failures of memory.

Sometimes plans or primitives will employ executives which, for efficiency perhaps, simply do not record all of this information. For example, the temporal order in which justifications are constructed is usually not recorded, although these justifications actually record actions taken by primitives. While it may be possible to introduce such temporal records in a serial computer, there is reason to suspect that the parallel computations which may ultimately be necessary (and which may be used by humans) will rule out having complete temporal records.

### 4.11 The Frontier

We partition the set of intentions (current or not) into three segments: the *past*, the *present*, and the *future*. The past consists of all the finished or discarded intentions, the present of all active intentions, and the future of all pending intentions. In addition, we further subdivide the future into the *frontier* and *blue sky*. The frontier consists of all enabled pending intentions, and blue sky all blocked pending intentions. The past thus contains all intentions that have been either discarded or, more commonly, carried out, the present all intentions currently being carried out, the future all current intentions yet to be carried out, the frontier those current intentions which can be worked on directly, and blue sky those intentions which depend on the successful completion or satisfaction of prior intentions. The terminology blue sky is meant to recall that the opportunity to work on blue sky intentions depends on everything going well, on no unforeseen circumstances arising which lead to the premature abandonment of the intentions due to impossibility or inappropriateness. We make these distinctions because the program normally acts only on the intentions on the frontier.

## 4.12 A Careful, Meta-Circular Interpreter

How does the program act on the basis of its desires and intentions? This question has many answers, for interpretation of the current mental state is an activity itself, and like other activities, can be performed in many ways. For example, the basic steps of acting on the basis of the desires and intentions are (1) pick a desire or intention to carry out, (2) pick some way of carrying it out, (3) carry it out via the selected means, and (4) repeat these steps. There are clearly many ways of going about these steps. One can be very careful about what one is doing and deliberate at length in steps (1) and (2), or one might just carelessly pick a task and tack at random, or something in between these extremes. As another example, one might choose to work for some while only on one intention and its subordinates to the exclusion of all other independent activities, for instance, exclusively pursuing thesis-writing and its subactivities to the exclusion of social and educational activities. In fact, this provides a way of viewing primitive programs as extremely specialized executives, executives which start with one intention and singlemindedly pursue it and its subactivities (although the subactivities of primitives are usually not explicit intentions but rather further primitive calls). Thus there is an extremely wide range of executives employed by the program, and the typical operation of each of these is to exercise control of the program's actions until it interprets an instruction to hand over control to some other executive.[62]

This section describes a very careful and general interpreter. This interpreter is particularly interesting in that it is a *meta-circular* interpreter, one written in the language that it interprets. In this case, the standard way to do things carefully is to plan them, and this interpreter, or TORPID as we will call it, follows this strategy by being a plan containing a set of plans, method statements, and policies for interpreting the current state of mind, and so plans how to carry out its own intentions. The heart of TORPID is the following plan, whose steps are outlined in Figure 10.

[62] This sort of approach to program executives is sometimes called continuation-passing style [Steele and Sussman 1976].

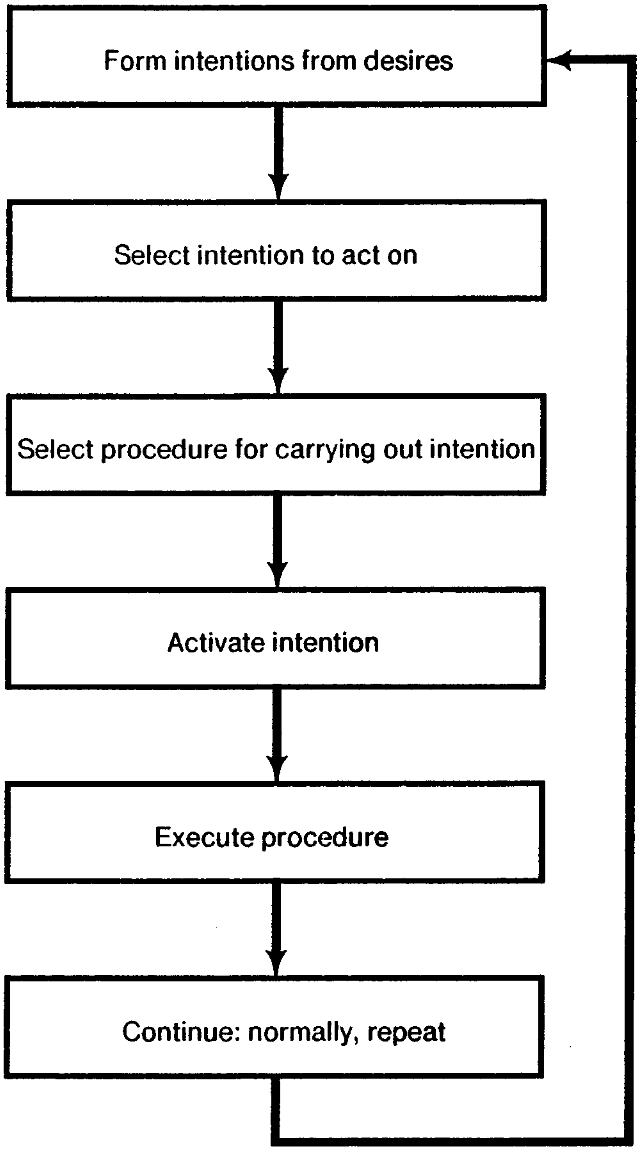

Form intentions from desires
Select intention to act on
Select procedure for carrying out intention
Activate intention
Execute procedure
Continue: normally, repeat

Figure 10

The TORPID Procedure

```
IN TORPID: (implicitly in all the following)

(DEFPLAN MACRO-TORPID                          ;Defplan, Choose, Aspect - explained below
 (INTENTION I-1 () () ((CHOOSE (ASPECT=AIM) (INTENTION=∃I INTENTION(I))) () ()))
 (INTENTION I-2 () (INTENTIONS) (FIND-FRONTIER-INTENTIONS () (INTENTIONS)))
 (ANTECEDES I-1 I-2)
 (INTENTION I-3 (INTENTIONS) (CHOSEN-INTENTION)
    ((CHOOSE (ASPECT=CHOSEN-INTENTION) (INTENTION=I-7)) (INITIAL-OPTIONS) (OUTCOME)))
 (INTENTION I-4 (CHOSEN-INTENTION) (METHODS) (FIND-INTENTION-METHODS (INTENTION) (METHODS)))
 (INTENTION I-5 (METHODS) (CHOSEN-METHOD)
    ((CHOOSE (ASPECT=CHOSEN-METHODS) (INTENTION=I-7)) (INITIAL-OPTIONS) (OUTCOME)))
 (INTENTION I-6 (CHOSEN-INTENTION) () (ACTIVATE-INTENTION (CHOSEN-INTENTION) ()))
 (ANTECEDES I-5 I-6)
 (INTENTION I-7 (CHOSEN-INTENTION CHOSEN-METHOD) () (EXECUTE-INTENTION (INTENTION) (METHOD)))
 (ANTECEDES I-6 I-7)
 (INTENTION I-8 () () (CONTINUE () ()))
 (ANTECEDES I-7 I-8))
```

Here we have used a syntactic macro to make a somewhat less verbose syntactic form for defining plans.[63] In DEFPLAN, one first specifies the name of the theory, MACRO-TORPID, and then in the body of the macro specifies the desires, intentions, and other parts of the plan with further syntactic extensions. The syntax for intentions specifies first the name of the intention in the plan, then the list of names of input variables, the list of output variable names, and finally the aim. The aim consists of the type of the aim theory, together with two lists of names. These should be names of parts of the aim theory, to be identified, respectively, with the lists of input variable names and output variable names of the intention to set up the keyword mapping of variables. In addition, the macro automatically sets up dataflow policies between all similarly named intention and plan variables, unless the names are mentioned in explicit dataflow policies.

What does this plan say? MACRO-TORPID's first step is to deliberate on things to do, to form intentions from some of its desires. This decision is formulated as an intention to choose aims (and, actually, variables as well, but that is left out for simplicity) for some unspecified intention. The intention is identified as a decision intention by the aim keyword "CHOOSE." The "ASPECT" statement indicates

[63] The exact details of this macro and syntax are yet to be worked out, but the main points should be clear. If something in the following seems underspecified, it is.

what part of the unspecified intention is to be filled in with a value. As its second step the interpreter finds the current set of frontier intentions, and names this with the plan-variable INTENTIONS. The third step is to choose one intention from this set and call it CHOSEN-INTENTION. This decision is formulated as a choice of a value for the variable CHOSEN-INTENTION of I-7. Fourth, the plan retrieves a list of methods relevant to carrying out the chosen intention, and calls this METHODS. Fifth, it selects one of these methods by using an intention to select a value for the variable CHOSEN-METHOD of I-7. The sixth step activates the chosen intention by changing its status. Seventh, it realizes the selected intention via the selected method. Eighth and finally, it continues interpreting.

MACRO-TORPID, to work as we have indicated, must be supported by a number of other plans, the appropriate method statements, and policies.

The first step of MACRO-TORPID relies on a careful deliberation procedure. The next chapter presents one of these. In this step, it is used to decide if any new intentions should be formed to pursue current desires.

The second step of MACRO-TORPID gathers up the current frontier intentions by means of a simple primitive program (omitted here) which scans the set of intentions for frontier intentions. (Alternatively, the actual implementation maintains a list of all frontier intentions, and modifies the list's contents when intentions and ordering policies are added and realized.) This primitive is declared to be the default method for this intention by a policy. Here we employ further syntactic macros to define Lisp functions as primitive concept attachments (DEFPRIMITIVE), to declare construct method statements for aims and procedures (DEFMETHOD), and to declare policies (DEFPOLICY) by giving the antecedent and consequent of their aim, the consequent being a list of instructions to be carried out (as the next chapter explains).

```
(DEFPRIMITIVE BASIC-FIND-FRONTIER-PRIMITIVE () (INTENTIONS)
 ...omitted...)
```

```
(DEFMETHOD BASIC-FIND-FRONTIER-METHOD
 (AIM [AIM T-2 INTENTIONS MACRO-TORPID])
 (PROCEDURE BASIC-FIND-FRONTIER-PRIMITIVE))
(DEFPOLICY BASIC-FIND-FRONTIER-DEFAULT-POLICY
 (IF ([AIM PURPOSE DECISION] =
    (CHOOSE-METHODS [AIM T-2 INTENTIONS MACRO-TORPID])))
 (THEN (DEFAULT BASIC-FIND-FRONTIER-METHOD)))
```

All of the following steps of MACRO-TORPID are carried out by similarly described primitives, which we will not give here, except for the last step of continuing execution. In this case, the default method for continuing execution is MACRO-TORPID itself.

```
(DEFMETHOD BASIC-CONTINUE-METHOD
 (AIM [AIM T-8 INTENTIONS TORPID])
 (PROCEDURE MACRO-TORPID))
(DEFPOLICY BASIC-CONTINUE-DEFAULT-POLICY
 (IF ([AIM PURPOSE DECISION] =
    (CHOOSE-METHODS [AIM T-8 INTENTIONS MACRO-TORPID])))
 (THEN (DEFAULT BASIC-CONTINUE-METHOD)))
```

This TORPID plan is all well and good, but how does the program get going in the first place? The answer is that it contains a primitive executive specially tailored for interpreting MACRO-TORPID. This executive is the following LISP[64] primitive program.

```
(DEFPRIMITIVE MICRO-TORPID ()
 (PROG (INTENTIONS INTENTION METHODS METHOD)
        (SETQ INTENTIONS (MICRO-TORPID-FIND-FRONTIER-INTENTIONS-PROCEDURE))
        (SETQ INTENTION (BASIC-CHOOSE-NEXT-INTENTION-PROCEDURE INTENTIONS))
        (SETQ METHODS (BASIC-FIND-INTENTION-METHODS-PROCEDURE INTENTION))
        (SETQ METHOD (BASIC-CHOOSE-INTENTION-METHOD-PROCEDURE INTENTION METHODS))
        (BASIC-ACTIVATE-INTENTION-PROCEDURE INTENTION)
        (COND ((PRIMOP? METHOD)
            (BASIC-INTENTION-EXECUTION-PROCEDURE INTENTION METHOD))
           (T (BASIC-INTENTION-REDUCTION-PROCEDURE INTENTION METHOD)))
        (MICRO-TORPID)))
```

MICRO-TORPID has roughly the same steps as MACRO-TORPID, but with the defaults of TORPID built into place. It calls the TORPID primitives directly, except for the decision of what intention to work

[64] Actually, SCHEME would be better. The recursive call of the last line would have to be replace by a loop for it to work in LISP.

on next, for which MICRO-TORPID uses a procedure which looks only for intentions resulting from an instantiation of MACRO-TORPID. Also, MICRO-TORPID does not deliberate on what to do because it only looks for intentions resulting from an instance of MACRO-TORPID.

Let us see how this works.

1. We start up the program by constructing an intention to CONTINUE and executing MICRO-TORPID.

2. Because there are no other intentions, MICRO-TORPID picks this intention as the next step, finds its default method, namely MACRO-TORPID, and reduces the intention to the new plan, a copy of MACRO-TORPID. MICRO-TORPID then begins work on MACRO-TORPID.

3. MACRO-TORPID's first step is to deliberate on what to do. For this it uses a careful deliberation procedure as described in the next chapter. This deliberation procedure finds possible courses of action by means of a policy to fulfill the desires if possible. The ordering policies between the desires, and other policies as well, provide reasons for and against these options. When this deliberation is finished, all options that have good reasons for them and none against them are used to form new intentions.

Intention formation seems to be ill-studied, to the best of my knowledge. The approach taken here is no more than an initial, and likely unsatisfactory, proposal for how this might be done. In MACRO-TORPID, normally all desires eventually are pursued by forming intentions to pursue their aims. This step is the means by which intention formation occurs. No intentions might be formed, or several might be formed, depending on what sorts of policies enter into the decision-making. For example, policies which reflect on the resource limitations implied by the program's current intentions might rule out forming any new intentions. Policies which reflect on the consistency of desires and intentions may rule out some desires but not others. Or at the other extreme, the program might find unchallenged reasons to pursue all its desires, and form intentions from all of them. This subject deserves more serious attention than I have been able to give it.

It might seem that this step could be combined with MACRO-TORPID's third step of picking an intention to carry out, but this cannot be, for two reasons. First, one can decide to pursue a desire, but not an intention. It makes no sense to intend to intend to do something. Second, if one deliberated about desires and intentions simultaneously, one would need values comparing desires and intentions, which also makes no sense. In fact, one way to compare intentions might be to compare the strengths of the desires they were formed from, if there were any, but intentions cannot be compared with desires directly.

4. MACRO-TORPID's second step is to find the frontier intentions. At this point, there are no frontier intentions, because the only other intentions are those in MACRO-TORPID itself, which are blocked for lack of input variable values. Thus when MICRO-TORPID retrieves and deliberates on methods for this intentions, it not only finds the default primitive, but also the following backup primitive.[65]

```
(DEFPRIMITIVE INPUT-NEW-INTENTIONS-PRIMITIVE () (INTENTIONS) ...)
(DEFMETHOD INPUT-NEW-INTENTION-METHOD
 (AIM [AIM I-2 INTENTIONS MACRO-TORPID])
 (PROCEDURE INPUT-NEW-INTENTIONS-PRIMITIVE))
(DEFPOLICY INPUT-BACKUP-POLICY
 (IF ([AIM PURPOSE DECISION] =
    (CHOOSE-METHOD [AIM I-2 INTENTIONS MACRO-TORPID])
        ∧ (BASIC-FIND-FRONTIER-INTENTIONS-PRIMITIVE,) = NIL))
 (THEN (CON (OPTION 'BASIC-FIND-FRONTIER-INTENTIONS-METHOD) (SL (INPUT-BACKUP-POLICY) ()))
   (PRO (OPTION 'INPUT-NEW-INTENTIONS-METHOD) (SL (INPUT-BACKUP-POLICY) ())))))
```

The backup primitive INPUT-NEW-INTENTIONS-PRIMITIVE queries the user for some intention to work on and waits for a reply. The backup policy leads MICRO-TORPID to select and execute this primitive for finding new intentions rather than the normal one which just looks at the frontier.

5. At this point, we enter construct some intention along with procedures for carrying it out.

[65] This should be done in some better way, such as reflecting on how to proceed as does NASL with reformulation intentions, but I have not attended to this problem yet. If done properly, we could just call MICRO-TORPID at the start and let it ask for the initial MACRO-TORPID CONTINUE intention. This primitive also shows the paucity of communication of the program with its environment. If all new information is gathered unconsciously by primitives or added by the user while the program's operation has been interrupted, then the program is unconscious of its environment. To have the program be conscious of its environment as well as merely self-conscious, it must have information about its sensory and effective mechanisms so that it can use its communication channels deliberately, rather than simply reacting to their automatic functioning.

We ignore the details of this, for our concern is primarily with watching TORPID.

6. The backup method now returns the new intention as the frontier. This frontier is recorded as the value of the plan variable INTENTIONS. Thus while executing TORPID, the program leaves behind records of the intentions it saw at some past step of interpretation. This is an important piece of historical information useful in the skill modification processes discussed later.

7. Next, MACRO-TORPID presents the frontier intention of choosing which intention to work on from INTENTIONS. MICRO-TORPID sees both this intention and the non-TORPID intention on its frontier, but restricts itself to working only on intentions stemming from instances of MACRO-TORPID, so works on MACRO-TORPID's third step.

8. MICRO-TORPID carries out MACRO-TORPID's third intention by the default method, which is a general deliberation procedure. MACRO-TORPID's third step is not one of forming any number of intentions, but rather one of deciding on a single value for an aspect of a current intention, namely the variable CHOSEN-INTENTION of I-7 in the current instance of MACRO-TORPID. The deliberation procedure sets up the frontier intentions transmitted through INTENTIONS as the initial options. It proceeds to find reasons for and against carrying out each of the intentions next. Finally, it decides on one, and attaches that value to the specified variable of I-7.

Actually, we have been needlessly redundant in MACRO-TORPID for the sake of clarity. The policies relating intentions that determine the frontier actually enter into this deliberation, so we can just as well dispense with step I-2 (and similarly, I-5) by beginning deliberation with a policy to make all pending intentions options, and then forming reasons for and against these options from the policies. This would also make unnecessary the complex system of justifications between intention statuses used to compute the frontier.

All in all, the deliberation procedure subsumes all the special case information mentioned above. The list of options of the deliberation record shows what intentions were considered at this time, and the list of considerations shows the extant policies. In addition, there may be other policies relevant to this decision besides the policies. In fact, some of the current ordering policies might be defeated by special-case policies, so the frontier as seen from looking just at the temporal ordering policies is not always completely accurate.

At any rate, the deliberation procedure in this case chooses the sole non-TORPID intention as its outcome, and so the plan variable CHOSEN-INTENTION is given this value.

9. Next, MACRO-TORPID retrieves a list of methods for carrying out the chosen intention and stores the list in METHODS. It then deliberates to find a method as the value of CHOSEN-METHOD. As above, the deliberation step actually subsumes the prior retrieval step.

10. MACRO-TORPID's next step performs some bookkeeping functions, primarily to change the status of the chosen intention to active.

11. MACRO-TORPID then has the step of realizing the chosen intention by the chosen method. It performs this realization either by executing the procedure selected as the method or by adding the method plan to the current state of mind. This intention of MACRO-TORPID forms the realization statement connecting the intention and its realization, obviating the need for the interpreter to make such a record specifically. In addition, since the ordering for the steps of MACRO-TORPID is a standard linear order, we also get historical ordering records between the realization records automatically.

12. Finally, MACRO-TORPID again presents the intention of continuing, and the process starts again. The connection between this intention and the new instance of MACRO-TORPID then forms the next part of the chain of historical order.

This concludes the example of TORPID's operation.

The program need not operate solely by using TORPID as the interpreter, but might use at different times a number of interpreters. In fact, the program can employ a slight generalization of MICRO-TORPID which records the desired records (or not) without going to the extremes of deliberation met in TORPID. The program can switch between "careful" mode and a normal heedless mode which does not record as much information by the following technique. Say that the plan to be executed is to carefully perform some action. Then the first step of the plan is to switch to the careful interpreter by means of a CONTINUE intention. The remainder of the plan then is executed by MACRO-TORPID. In addition, the plan sets up the return mechanism by having its last step be one which temporarily changes the default method for CONTINUE to the standard interpreter (or whatever executive is desired next). Since MACRO-TORPID will be careful in reducing its continuations, it will resume with the specified executive rather than MACRO-TORPID.

# CHAPTER 5

# DELIBERATION

How do we ever manage to make decisions? The overwhelming fact of our lives is the dilemmas and near dilemmas that confront us, the difficult decisions which force us to sacrifice one hope for others. We are constantly torn between seemingly incompatible principles of action. To decide what to do we must reconcile these principles to choose the moral, the kind, the expedient, or the comfortable thing to do.

Dilemmas and difficult decisions involve reasons for conflicting courses of action, reasons formulated in disjoint vocabularies and value systems. To resolve dilemmas, whether they be mighty or mundane, the decision-maker needs to reflect on these conflicting reasons, to consider what value-systems each reason belongs to, and to judge which reasons take precedence over which other reasons. We must be content to make these precedence judgements in a case by case fashion without absolute principles or reductions relating the disjoint value-systems like "Any mora; reason takes precedence over any economic reason." This chapter explores such a decision-making method called *reasoned deliberation* in which the deliberation procedure can reflect on the reasons for and against courses of action, and make judgements about these reasons specific to each particular decision and its circumstances.

The basic idea of reasoned deliberation builds on the mechanisms developed in the preceding chapters. The program first formulates its intention to make a decision as a *decision intention.* It then makes the decision by executing a *deliberation procedure,* which is retrieved as a method for carrying out the decision intention. There are many sorts of deliberation procedures corresponding to the many sorts of decisions to be made, but the typical general-purpose deliberation procedure constructs a set of *options,* a set of relevant policies (called *considerations*), and a set of *reasons.* The policies retrieved from the current set of intentions as being relevant to the decision are carried out to construct reasons for and against the options, and to augment the set of options with new options. However, each reason constructed by carrying out a policy for a particular decision is a non-monotonic assumption. Each policy represents an intention to reason in a certain way, and this intention is satisfied by constructing the appropriate reasons. The policy's putative effect may fail to be realize because the policy's application in a particular decision may be defeated by other policies concerning special cases, exceptions, or preferential relations among types of reasons. The deliberation procedure reflects on each new reason to find further policies relevant to the new reason. These further policies might construct reasons against the original reason. Since the reasons are non-monotonic assumptions, these new reasons defeat the original reason, defeating the application of the original policy in this particular case. Of course, these defeating reasons can in turn be defeated. Finally, the deliberation procedure reflects on the entire set of reasons to decide whether to make a decision on the basis of the constructed reasons, to postpone the decision, to deliberate further, or to reject the decision.

Reasoned deliberation plays an important role in the operation of the program. For example, in some cases this sort of reflective decision-making is used by the interpreter to form the intention to pursue some desire, to select which intention to carry out next, or which method to use in carrying out the selected intention. In Chapter 6 we will indicate further applications of reasoned deliberation in deciding how to revise or modify the program's sets of beliefs, concepts, desires, intentions, values, and skills.

Of course, not all decisions are made by procedures of the complexity outlined above. In many cases, one has decided in advance how one will make a type of decision in certain circumstances, and when such occasions arise, one simply executes that procedure. These prior decisions with their built-in presuppositions can fix assumptions or the use of particular policies in a specialized decision procedure, so that the special-case procedure need not be as complex in operation as the general procedure which has to retrieve and decide how to apply assumptions and policies on the spot. In fact, this deciding how to decide is a common activity. Since the plans and primitives of the program are really specialized executives making certain types of decisions, the choice of which method to use in carrying out some intention constitutes a decision about which further decisions to make and how to make them. This is clearest in the case of deciding what procedure to use in carrying out a decision intention. Since the library of procedures is organized hierarchically, these decisions about decisions are made in a similar hierarchical fashion. This can be viewed as analogous to the practice in large corporations and other organizations in which each level of management makes some decisions itself, but spends a good bit of effort in deciding how to delegate decisions, that is, who should make the lower level decisions. Eventually someone makes decisions about concrete matters, but his position is the result of many prior decisions about who should make decisions.

## 5.1 The Variety of Decisions and Ways of Making Them

There are many different sorts of decisions one makes, and different decisions call for different procedures for making them. For example, we can imagine different procedures for buying a can of tuna, for buying a car, for selecting one's job, and for thinking of what to do tonight.

Example 1: Grocery shopping. When shopping for some item in a supermarket, say a can of tuna, my standard procedure is to buy the same brand and size as I bought before, unless prices seem to have changed or there is a sale on some comparable item. On some occasions, my use of a product previously has made me dissatisfied with it, so I do not even bother to check for a sale, but rather use a completely different procedure from the start: to compare all the available brands and sizes, their reputation and looks and specifications, and choose something different from before, even if just to experiment. A further different way of buying a product, which I only employ in exceptional circumstances, is to go in, compute which product is cheapest in unit price, and buy that.

Example 2: Buying a car. In contrast to grocery shopping, buying a car is never routine, but is always a major decision. This is reflected in the ways of choosing what to buy in the extra care, prior experimentation, and time allotted to making this decision. Where I might be content to experiment on my own to find my preferred brand of tuna, I am likely to begin a search for a car by talking with friends to get their experiences, by reading car magazines and *Consumer Reports*, by making the subdecision of whether to buy a new or used car, and by visiting several vendors, kicking tires, and making road tests of several models.

Example 3: Choosing a job. Assuming one's financial situation leads one to take a job, one faces different problems in choosing one than in buying tuna or cars. Here any prior experimentation with the job becomes part of the job, so one cannot perform prior experimentation in the same way as with cars. Instead, the major bases are, for example, one's self-analysis of what one likes, what opportunities are to be had or made in different lines of work, and how one's favorite role models earn a living. These deliberations sometimes involve considerations which in essence reject or postpone the choice, such as choosing a deliberately temporary occupation to be rethought later or to continue schooling.

Example 4: What to do tonight. This question is much like the question of what to do next that the interpreter faces at every step of its process. However, for humans, deliberating on this question usually involves not just a selection between current intentions, but invention of options by, for example, looking in a newspaper or asking friends to see what is in town, or walking through a bookstore or library to see if there is anything interesting to read, as well as thinking about standard possibilities like visiting friends, museums, ice-cream parlors, or coffeehouses.

These examples illustrate several different procedures for making specific sorts of choices. The actions involved in these procedures range from making simple arithmetical calculations to running extensive physical tests on machinery to mental tests of oneself or others. While one might use an abstract deliberation procedure in novel situations, efficiency dictates that we employ special purpose procedures in routine cases. Such specialization might restrict the options involved so that we do not waste time searching for unusually creative ways of procuring tuna (placing wantads in a paper, for example), or might restrict the sorts of reasons we take for choosing which option, such as computing the cheapest unit price instead of physically inspecting a fishing fleet or cannery.

Aside from classification by types of decision, the major general classification of decision procedures is whether the procedure *chooses* or *deliberates.* This distinction is traditional in discussions of decision-making, and attempts to draw an (admittedly hazy) line between reflective (deliberation) and non-reflective (choosing) decisions. For example, at a party someone offers you a tray full of drinks, and you pick one without thinking about it. This is called choosing, as it did not involve considering all the option and reasons in detail. One might also perform choosing if one has sequential preferences for ice-cream flavors, and one always orders a flavor by checking the parlor's list of available flavors and taking the one highest on one's own list. However, one is deliberating if one picks the drink after first considering "Should I have another? Who is driving home?" or picks the flavor by trying several samples and deciding which one is the most intriguing. Thus in choosing, one follows a routine procedure which has only fixed variability, or whose variables depend on the external world and not on one's store of guidelines. Deliberation, on the other hand, varies with what principles one has adopted and retrieves upon thinking about the question.

We only briefly discuss choosing, about which we just recall the earlier suggestion that choosing procedures are programs "compiled" by fixing in advance the policies to use as implicit assumptions. For example, one might employ a policy in buying tuna which computes the unit prices and constructs a reason for the tuna with the lowest unit price. If one decides in general to act on this policy alone, one can take the policy and computation code used to carry it out to produce a procedure which simply makes the computation and justifies its answer, to be used instead of the general procedure which would have had to retrieve, apply, and defend this policy.

The remainder of this chapter concentrates on deliberation.

## 5.2 Decision Intentions

Decisions are mediated through *decision intentions*, which are intentions to make certain decisions. Decision intentions are just like other intentions, except that their aim is to make some decision. Decisions are all of the form of choosing between alternate actions, although the actions may be mental, such as believing something, as well as physical. Thus the aims of the decision intentions are all of the form (CHOOSE aspect-name action-description), to be interpreted as the intention to find a value for the aspect-name of action-description. But action descriptions are really potential intentions, so the aim of the decision intention really reads as (CHOOSE aspect-name intention). From the hierarchy of aims of intentions and the related hierarchy of intentions, we so derive yet another hierarchy, that of decisions. In the aim of a decision intention, aspect-name can be any term referring in the intention theory. If the aspect name is AIM, then the aim of the decision intention is to find an aim for the intention, that is, what to do, the most general question of action. If the intention theory already has an aim, then the aspect name might refer to some term in the aim theory or in one of its subtheories. Thus we might have the following hierarchy of decisions corresponding to a hierarchy of intentions described by a hierarchy of aims.

**DECISIONS**

**What to do**

| | |
|---|---|
| What to buy | Where to go |
| What food to buy | Where to go this summer |
| What tuna to buy | Where to go in Disneyland this summer |

**INTENTIONS**

**Do THING**

| | |
|---|---|
| Buy THING | Go PLACE WHEN |
| Buy FOOD | Go PLACE this summer |
| Buy tuna | Go PLACE in Disneyland this summer |

**AIMS**

**AIM of intention**

| | |
|---|---|
| Object of (Buy) AIM | Location of (GO) AIM |
| Object of (Food-Buy) AIM | Location of (GO with WHEN=this summer) AIM |
| Object of (Tuna-Buy) AIM | Location of (GO with WHEN=this summer AREA=Disneyland) AIM |

## 5.3 Deliberation Records

We introduce the convention that the important information concerning a deliberation is recorded in a *deliberation record.* A deliberation record is a theory constructed by deliberation procedures, and can be thought of as a record of the basic variables common to all deliberation procedures along with their values in the deliberation at hand. A deliberation record theory has several parts: a purpose, a list of options, a list of considerations, a list of reasons, a list of reflections, and an outcome. We explain these parts in turn. See Figure 11 for a picture of how these pieces of information are related.

The *purpose* of the deliberation record is simply the decision intention being carried out by the deliberation procedure. Deeper purposes or reasons for why the program is making the decision are found by pursuing the reasons for this decision intention.

The *list of options* lists the objects being decided among, the possible values for the aspect of the intention being deliberated about. The interpreter, for example, makes decisions whose options are the desires to pursue next, the possible values (or identities) of some variable, the methods retrieved for some intention, or the possible revision of belief which restore consistency. They need not be exclusive in any sense.

The *list of reasons* lists the reasons for and against choosing the several options. In addition, the reasons themselves are treated as things to reason about, so the set of reasons also contains the reasons for and against the reasons as well. All reasons are recorded as explicit, defeasible justifications as described in Section 3.11. Reasons for and against options are made as justifications for, respectively, statements of the form `PRO(option)` and `CON(option)` in the theory describing the list of options, where `option` is the name of an option in that theory. Reasons for and against other reasons are made justifications supporting or defeating the other reasons. RMS then determines the status of the arguments comprising these reasons.

The *list of considerations* lists intentions to apply the policies retrieved as relevant to the

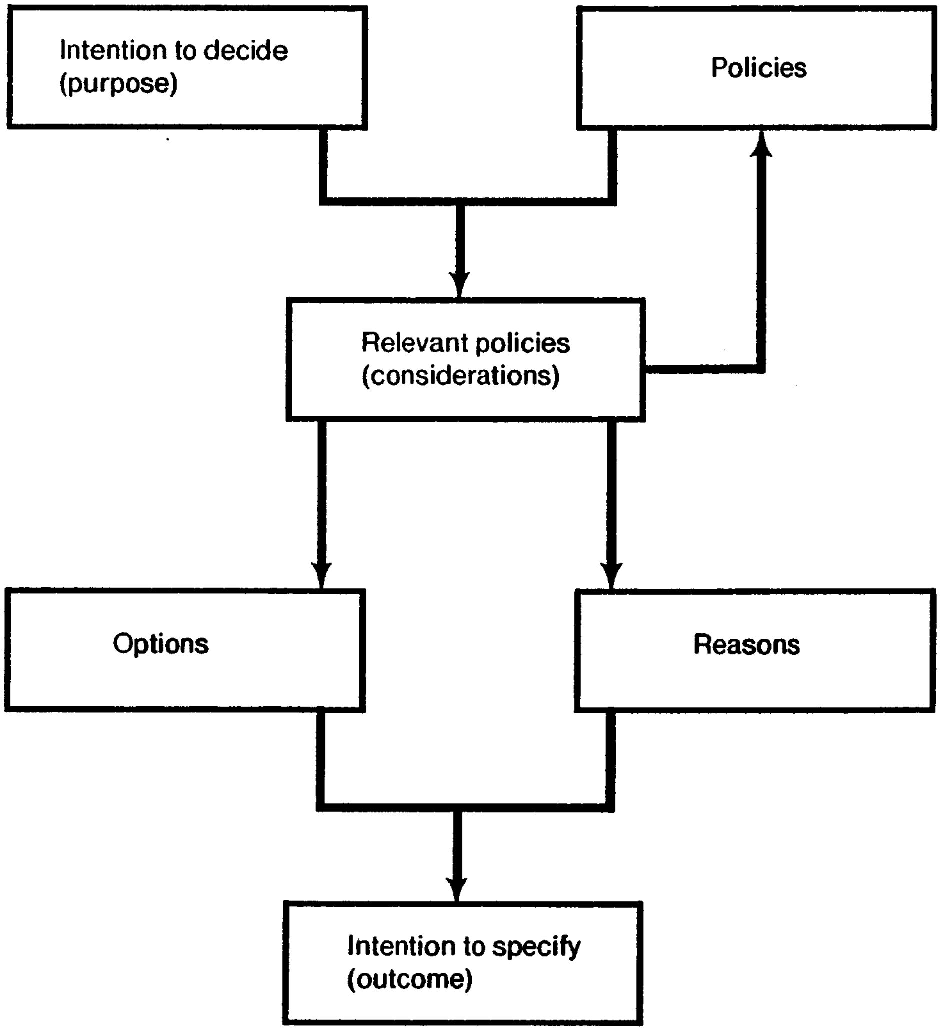


Figure 11

Information Flow in Deliberations

deliberation record's purpose. Typically these policies produce one or more reasons in the set of reasons, or add to or otherwise modify the set of options. Considerations are kept separate from the reasons they produce, because policies may be relevant even if they lead to no reasons. One sometimes says something is a consideration even if it implies no reason or option in the particular situation at hand, but does when in slightly different situations.

The *list of reflections* of the deliberation record lists the higher-level deliberation records created by deliberations reflecting on the progress of the decision intention. We will explain these reflective deliberations soon. These reflections are not used by the deliberation record itself, but rather aid the reflecting deliberations in accessing the results of previous reflecting deliberations.

The *outcome* of the deliberation record is the chosen option, if and when one is chosen.

These pieces of information are represented as attachments to the terms PURPOSE, OPTIONS, CONSIDERATIONS, REASONS, REFLECTIONS, and OUTCOME in the deliberation record theory. PURPOSE is attached to the decision intention, and this attachment is justified in terms of the realization record of the deliberation procedure carrying out the intention. The outcome, when it is found, is attached to OUTCOME with a similar justification. OPTIONS, CONSIDERATIONS, and REASONS are all attached to theories whose languages include the numerals 0, 1, 2, etc. Each of successive option, consideration, and reason is attached in these theories to one of these numerals, in the order of their discovery, thus recording something of the temporal order of the deliberation. Options and reasons are constructed by policies in the set of considerations, and their justifications reflect the policies and other facts used in applying the policy. Considerations are attachments to intentions (as explained in the next section), and these attachments are justified in terms of the retrieval procedure used and the data the retrieval procedure accesses, such as the purpose of the deliberation record and other beliefs. Note that the reasons constructed by a consideration depend only on the policy, and not on how it was retrieved.

### 5.4 Policy Execution

Policies are intentions with hypothetical aims, and as such, cannot be carried out directly. Instead, when conditions arise which might satisfy the condition of the hypothetical aim, the policies are used to form further intentions, intentions whose aims are to check if the policies are indeed applicable in the current circumstances, and if so, to carry out the consequential actions specified in the aims of the policies.

Policies are retrieved by procedures which scan the current set of intentions for policies whose aims have a condition subsuming the aim of the decision intention. Actually, the details of how this should be done have yet to be worked out, for the conditions specified by policies can include information other than that of the decision intention's aim, such as current beliefs, other intentions, etc. However, the retrieval procedures are not burdened with determining actual applicability of the policies, but mere relevance. This lesser requirement might be discharged by using explicit statements that certain classes of policies are relevant to certain classes of decisions, or by other means, but we leave this question to be answered by future study.

For each relevant policy retrieved, a new intention is formed. The new intention's aim is to *apply* (or *consider*) the policy in the current circumstances. The intention is made a subordinate of the decision intention, and is justified in terms of the policy, the decision intention, and the relevancy procedure. The new intention is added to the list of considerations of the deliberation record.

Each consideration intention is carried out as usual by the interpreter. A consideration may be carried out by any of several sorts of procedures. The common function of these procedures is to first check if the policy is actually applicable in the current decision, and if so, to carry out the policy's consequent instructions. These *application procedures* differ primarily in how careful they are in checking applicability and in carrying out the consequent instructions.

The default procedure for applying policies is a primitive which acts as follows. It first checks to see if the policy is applicable by applying to the policy's aim's antecedent a standard procedure for evaluating whether a logical formula holds in the current state of mind. For example, the FOL evaluator [Weyhrauch 1978] might be used. This procedure need not be perfect, for the default procedure is intended only for use in the simple routine cases. Since the policy's aim's antecedent is just a logical formula expressing some condition of the program's state, this test results in answers of either "it holds," "it doesn't hold," or "can't tell." Whatever the answer, a statement to that effect is recorded in the deliberation record, justified by the information and procedure used in the evaluation. This might permit later reconsideration of a policy whose applicability could not be determined earlier for lack of information. If the policy is inapplicable, or if its applicability cannot be determined, no further action is taken. If the policy is applicable, its consequent is interpreted as a sequence of instructions for actions to be taken. The vocabulary of these instructions is given in Section 5.6. They are carried out immediately by calling other primitives.

If a policy is not routine and deserves more careful treatment than this, other, more complex application procedures can be supplied to override the default application procedures. The care witii which policies are applied can be increased in many different ways. We sketch two of these.

A policy might be applied by carefully checking applicability and routinely executing its actions. That is, the applicability procedure is a plan of two steps. The first step is an intention to determine whether or not the policy is applicable. By making this step an explicit intention, the full power of the reasoner can be applied to carrying it out, rather than relying on a fixed and strongly limited evaluation procedure. The second step of the plan is an intention to act on the answer determined. This step is carried out by a primitive which acts like the default procedure, checking the answer and if it is that the condition holds, then calling primitives to carry out the policy's consequent instructions.

Another way to increase the care with which a policy is applied is to treat the consequent of the policy's aim as a plan. In such an applicability procedure, if the policy is applicable, then all of the instructions would be converted into new intentions and added to the current state of mind.

Policies might be applied by a combination of these refined procedures, or by yet other refinements.

## 5.5 Policy Applicability

Conditions of applicability typically refer to the superiors and other reasons for the purpose of the deliberation, to other intentions (such as the brothers of the purpose), to current beliefs, their reasons, and to the reasons and state of the arguments for and against the options in the deliberation record. For example, a policy to hold doors for ladies might be applicable only if the program currently believes it is near a door and whether there is a lady approaching. A policy not to act for chivalrous reasons might be applicable only if one of the reasons in the deliberation record is a repercussion of policies having to do with chivalry.

We again digress briefly to discuss deontic logic. We previously mentioned how retrieval of methods for carrying out intentions is related to the question of what commands or obligations are entailed by a command or obligation. Another question addressed by deontic logic is when commands or obligations can be inferred from beliefs together with previous commands or obligations. This is closely connected with the question of policy application. Policy application involves inferring a number of intentions (commands, obligations, etc.) from beliefs, intentions, and other aspects of the current state of mind. However, our approach makes this question trivial in principle, one purely of the validity of a logical statement about the current state of mind. Many deontic logics are complicated by the need to account for the defeasibility of reasons produced by policies. Our treatment suggests that this should not complicate the inference of intentions from policies, but should be separated into the treatment of the reasons constructed in carrying out these derived intentions.

## 5.6 Policy Actions

The actions of a policy either add new intentions as subordinates of the decision intention, options to the list of options, considerations to the list of considerations, or reasons to the list of reasons. We describe a few of these sorts of actions which form an initial vocabulary for decision-making activities.

The first sort of action is that of constructing a new subordinate of the decision intention. Subordinate addition is done with the command `(SUBORDINATE intention justification)`. The `intention` is the theory describing the intention to be added to the current state of mind. The justification is the justification to be used for the new subordinate. The justification usually mentions the policy, the application procedure, the realization record of the deliberation procedure, and any beliefs or other items used in determining applicability of the policy.

The second sort of action is that of adding new reasons to the set of reasons. One can add reasons either for or against either options or reasons. We write these sorts of actions as `(PRO {options/reasons} justification)` and `(CON {options/reasons} justification)`. In these and the following actions, options and reasons are referred to by their names in the lists of options and reasons, which are picked up by the condition of the policy.

Another action on the set of reasons is `(PREFER O {X ... Z} justification)`, where each of O, X, ..., and Z are options. Preference is translated as "Any good reason for O is a reason against any of X, ..., Z," so that a lesser option will have a good reason against it as long as a good reason holds for the preferred option and no special exceptions are being made (for example by some other policy reasoning against the preference statement). Preferences add a new policy to the list of considerations and to the current set of intentions whose aim is to reason against any reason for the lesser options (using CON above) whenever a reason for the preferred option is found.

A related action is `(DEFAULT option justification)`, which means that option is to be the default outcome. This is interpreted by giving the option a PRO (as above), and then to use any good reason for any other option in a reason to defeat this pro reason. This similarly is implemented by constructing a new policy.

A further action along these lines is `(BACKUP {O1, O2 ...} justification)` which is the policy to make O1 the default, and to make On+1 the default if On is defeated.

One might restrict the set of options, by providing a reason against any options not in the restriction. We say this with either `(RESTRICT {X ... Z} justification)`, or `(DECIDE-BETWEEN {X ... Z} justification)`, where X, ..., Z are options. A preferential restriction, `(PREFERABLY-RESTRICT {X ... Z} justification)` or `(PREFERABLY-DECIDE-BETWEEN {X ... Z} justification)`, uses any reason for any restriction option as a reason against each outside option. These also construct policies.

We add new options with `(OPTION X justification)` or `(OPTIONS {X ... Z} justification)`, where here X, ..., Z can be any objects of the sort required by the purpose.

An action operating on both reasons and options is `(REPLACE {W ... X} {Y ... Z} justification)`, where each of W, ..., X, is an option and Y, ..., Z can be anything, and are added as options. This means to replace the former set of options by a new set of options by preferring each of the replacing options to each of the replaced options. However, no new reasons for the replacing options are constructed. The action `(COMBINE {X ... Y} Z justification)` prefers Z to X ... Y and constructs a PRO reason for Z in terms of the policy and the reasons for the combined options. This is useful, for example, when reformulating options along a new dimension, when some options are each partly right and partly wrong, and a synthesis is possible which retains the good parts and discards the bad parts. This sort of case crops up very frequently when options are suggested on the basis of only a part of the problem. For example, when deciding what textbook to buy for some class, one might think of one book which is relevant for part of the class's charter, and another which is good for another part, but might then discover that some book covers both of these parts (such as the one written by the class's instructor).

## 5.7 A Very General Deliberation Procedure

In this section we present a deliberation procedure of considerable generality. Few situations call for as general a procedure, principally just novel situations and important decisions.

The procedure is, in essence, just that of repeatedly retrieving a relevant policy, carrying it out, and then reflecting on the results until the judgment is made during reflection to halt with a decision. This is of course a very cautious way of proceeding, and very time consuming, but sometimes this is necessary.

### 5.7.1 The Deliberation Plans

We first sketch the structure of the procedure as a set of informal plans, and then discuss its operation in detail using these plans as the framework. Figure 12 displays the basic steps of the plans.

```
DP-1:   Input: PURPOSE             Output: OUTCOMES
        1. Scan the set of intentions for relevant policies. For each one
           construct a new intention to consider it as a subordinate of
           PURPOSE, and add it to the list of considerations.
        2. Perform DP-2.
        3. Policy: Prefer step 2 (DP-1.2) to all the new subordinates of
           PURPOSE just constructed.

DP-2:
        1. Reflect carefully on what to do next (select the aim of step 2 (DP-2.2)).
                Options: Delay, Reject, Decide, Continue
                        Delay: Prefer non-DP tasks to DP ones until "later"
                        Reject: Abandon (defeat) PURPOSE
                        Decide: Set OUTCOMES, abandon unfinished subordinates
                        Continue: Perform DP-3 for one of the pending subordinates
        2. __________.  (Filled in by step 1 (DP-2.1).)

DP-3:   Input: SUBORDINATE
        1. Perform DP-4 for SUBORDINATE.
        2. Perform DP-2.
        3. Policy: Prefer step 2 (DP-3.2) to all original (DP-1) considerations.

DP-4:   Input: SUBORDINATE
        1. Carry out SUBORDINATE.
```

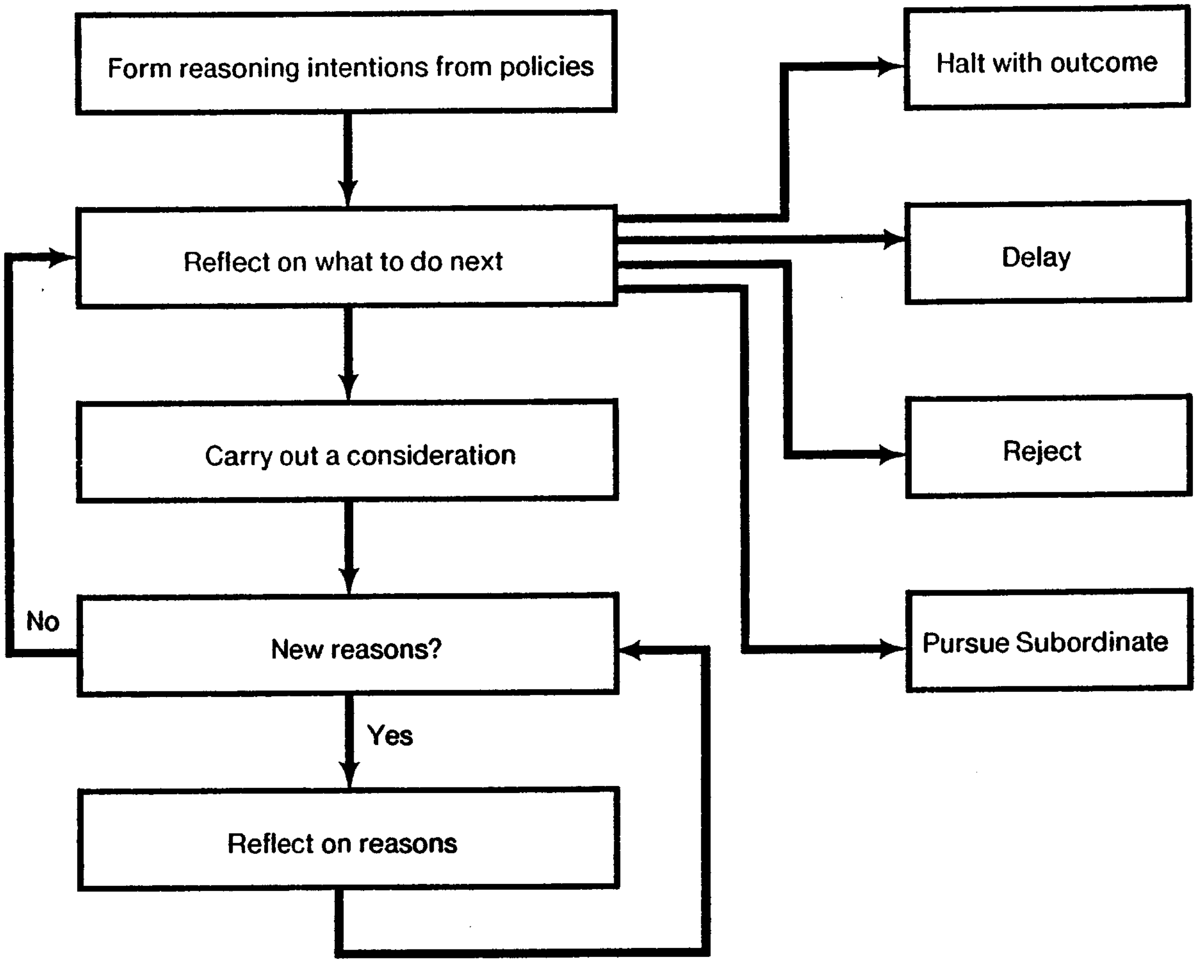

Form reasoning intentions from policies
Reflect on what to do next
Carry out a consideration
No
New reasons?
Yes
Reflect on reasons
Halt with outcome
Delay
Reject
Pursue Subordinate

Figure 12

The Deliberation Procedure

```
2. Scan the set of intentions for relevant policies. For each one
   construct a new intention to consider it as a subordinate of
   this step (DP-4.2) and add it to the list of considerations.
3. Policy: Prefer all intentions constructed in step 2 (DP-4.2)
          to all other DP intentions.
```

This deliberation procedure divides into two major aspects: the first-order reasoning, and the second-order reasoning. The second-order reasoning reflects on the first order reasoning to decide how to proceed with the decision-making process. We discuss each of these separately.

### 5.7.2 First-order Deliberation

1. *Create the deliberation record:* The first step towards making the decision is to construct a deliberation record, whose purpose is the decision intention being worked on by the deliberation procedure. If the intention (rather, the plan of which it is a part) also specifies initial options and defaults, these are entered into the deliberation record as well with justifications mentioning their source.

2. *Retrieve policies:* The second step is to search the set of intentions for relevant policies, using the purpose of the deliberation record as a means of determining relevancy.

Each policy retrieved adds a new intention to the set of considerations with the relevancy procedure and its arguments in its justification. The list of considerations will be scanned in Step 4 to carry out these policies one at a time. The new intention is that of applying the policy in this decision.

3. *Reflect on how to proceed:* The deliberation procedure is a UNTIL-REPEAT loop, repetitively considering policies until the decision is made to stop. This step asks the UNTIL question about how to proceed. It is the intention to reflect on the current progress of the deliberation and to decide whether to make a decision, to continue deliberating, or several other possible courses of action. In one sense, this step is much like the ordinary step of the interpreter of deciding what to do next, except that this decision is to be made relatively carefully itself. Its aim is not simply that of selecting one intention over another, but rather that of selecting between some intentions (the considerations and other

subordinates) and some possible but not actual intentions, that is, courses of action yet to be made into intentions by the deliberation procedure.

The interpreter sees at this point a frontier including this reflection intention, the unrealized subordinates of the original decision intention, and any other independent intentions, and it chooses one of these to work on. However, the deliberation procedure has set up policies to guide the interpreter by preferring the reflection intention to any other subordinates of the purpose. This preference will not be overridden by the decision intention or any of its subordinates, but might be overridden by independent intentions that have higher priority than further deliberation.[66]

At any rate, the third step is to invoke a second-order deliberation procedure to consider the problem of how to proceed with the original decision. As in first-order deliberation, the actions of the second-order deliberation procedure are to first create a deliberation record and then deliberate in that deliberation record. We postpone description of these steps for the next subsection, and proceed now with the rest of the :irst-order deliberation steps.

4. *Apply one policy:* The next step is to carry out an unrealized consideration as selected during reflection. The interpreter retrieves application procedures for carrying out the policy, selects one,[67] and executes it if it is a primitive, or added to the current state of mind if it is a plan. In the latter case, it is given priority over all other DP-related activities.

Alternatively, the previous reflection may have selected some non-consideration subordinate of the purpose, and in this case, that subordinate is carried out.

Part of what is properly second-order deliberation is built into the policy actions in the following way. If the actions add new options, the deliberation procedure retrieves and forms considerations for all policies relevant to the new option and the purpose, but does not carry them out yet. However, if the

[66] I do not specify how this selection is made. I expect that it is normally much simpler than the decisions made by the careful procedure.

[67] Again, I have not worked out in detail how this choice is made.

actions add new reasons to the set of reasons, then the deliberation procedure retrieves and forms considerations for all the policies relevant to the new reason and the purpose, and then carries out each of these new considerations the same way. This process of reflecting on new reasons continues until no more reason-relevant policies can be found.

Does this uncontrolled iteration always halt? If things are properly organized, yes. This can be seen by a simple argument. The conditions of these reason-reflecting policies are all basically of the form "If the decision is about X and a reason R of type T has been found for or against {a reason Ri of type Ti for or against}* an option O", where the starred, bracketed phrase may be repeated any number of times. That is, successively retrieved policies refer to successively longer arguments debating some option. Therefore, unless the system has infinitely many policies, this reason-reflection iteration must terminate.[68]

5. *Repeat:* The deliberation procedure now keeps repeating steps 3 and 4 until the decision is made to halt deliberation in one of the ways described in the next subsection.

### 5.7.3 Second-order Deliberation

1. *Construct the second-order deliberation record:* The purpose of this deliberation record is the second-order decision intention. This deliberation record is also added to the list of reflections of the first-order deliberation record.

2. *Engage in second-order deliberation:* The second-order deliberation procedure retrieves and forms considerations from all policies relevant to the second order decision. It then carries out each of these intentions, reflecting on each new option or reason to find newly relevant policies, but without reflecting on how to proceed. That is, these considerations are simply carried out one after the other,

[68] Of course, this "proof" has holes, but further investigation requires a working program and concrete examples. I do not foresee any serious difficulties.

barring interruptions from independent intentions, until all considerations have been realized and no more can be retrieved. We need not fear non-termination because of the limited and non-constructive nature of the policies applicable to the second-order decision.[69]

### 5.7.3.1 Second-order Options

There are a number of standard policies for this second-order deliberation. Some of these construct options and reasons standard in every second-order deliberation, others construct other options and reasons of sorts standard in every second-order deliberation, and yet others construct decision-specific options and reasons. The standard options are as follows.

*Option A*: Delay further work on the decision in favor of working on other intentions. This means to retain the original decision intention as an active, in-progress intention, whose execution will be resumed later. Taking this option means halting second-order deliberation after adding a policy which will preferably restrict the next step taken by the interpreter to some activity unrelated to the decision. Of course, there is a wide range of types of delays, from just avoiding the decision for one activity, to avoiding it until many other activities have been undertaken, to avoiding it until all other activities have been finished, to avoiding it until certain information is discovered. Formulating this sort of vocabulary is an area for future study.

*Option B*: Reject the decision. This means to discard the first-order decision intention, to defeat the intention to make the decision.

The options and policies of standard sorts are as follows.

*Option C*: Halt deliberation by deciding on the currently best option as the outcome. This

[69] An interesting direction for further exploration of these ideas is to develop a modification of this procedure so that the second-order deliberation procedure is the same as the first-order procedure. This would be a completely uniform, arbitrarily reflective deliberation procedure. Some sort of termination policies would be needed, or perhaps the default of halting rather than further reflection once the second level was reached.

means both setting the value of the plan variable for the outcome, and also defeating all unrealized considerations. This option is created by a policy that computes which first-order option has a good "overall" reason, plugs it into the form of this option, justifies this new option, and then creates a reason for this option, the reason being that the selected "overall" reason is a good "overall" reason.

When second-order deliberation decides to terminate the first-order deliberation by taking some first-order option as the outcome, it does so by finding some good reason "all things considered." There are several ways of interpreting what this means, and the one which we adopt here is that in the current set of reasons as interpreted by RMS, the selected option has a valid pro reason and no valid con reasons. The second-level reason for this second-level option actually comes in two forms, those in which the option is picked because it is the only such option, and those which pick the option randomly from several such options. These will be explained shortly.

In some cases, the deliberation procedure can return several outcomes rather than just one. The different restrictions are enforced by second-order policies about multiple "good" options. There can be a policy to return them all (as in deliberating on which desires to pursue), to pick one randomly (as in selecting the intention to carry out next), or to force just one outcome. This last restriction could be effected by a policy which defeats against each option on the basis of good reasons for any other options.

*Option D*: Continue deliberation by carrying out consideration intention I. An option of this form is created for each unrealized consideration I, and decision-specific policies may provide the option of reconsidering some previous policy. Reconsideration amounts to reapplying all of the relevant considerations and looking for further relevant policies and other new information.

*Option E*: Continue deliberation by carrying out non-consideration subordinate I of the decision intention. An option of this form is created for each unrealized subordinate I of the decision intention.

*Option F*: Reformulate the decision as I, that is, abandon the current decision intention, add the new intention I, and resume interpretation, which will eventually work on I afresh. This sort of option is never constructed by a general policy, only by domain-specific policies. Option B is the domain-independent form of this option. Option F is meant to cover the case in which thinking about one question leads to the discovery that the presuppositions of the decision are wrong. For example, one is trying to decide on an outline for a paper, and realizes that the important question is not about which organization is best, but about who is the intended audience of the paper. One then discards the active intention to decide on an outline, only later forming a similar intention after the audience decision has been made.

### 5.7.3.2 Second-order Policies

Along with these standard options, the standard second-order policies construct a number of reasons. These reasons for and against the second-order options involve a number of factors, including PC reason analysis, completeness information, compatibility information, time and resource pressure, and others. This subsection explains some of these sorts of factors and the policies involving them.

*PC reason analysis* classifies the options into four sets; PNC, containing those options with a valid pro reason and no valid con reasons (that is, those options O with the statements PRO(O) *in* and CON(O) *out*); PC, containing those options with both valid pro and con reasons; CNP, containing those options with a valid con reason but no valid pro reasons; and NPNC, containing those options with no valid reasons pro or con.

PC reason analysis is by itself insufficient for making decisions. The naive policies involving it alone might read as follows.

```
POLICY-1: If PNC contains exactly one option, take that as the outcome
          of the first-order deliberation record.

POLICY-2: If PNC contains more than one option, pick one randomly as the
          outcome of the first-order deliberation record.
```

However, with the deliberation procedure as we are outlining it, these policies are flawed, as there is no guarantee that more than one option has been considered, so that these policies might lead to an overly hasty decision. To remedy this problem, these policies must be modified to take the history of the deliberation into account. For example, Alfred P. Sloan Jr. refused to allow the GM executive officers to come to a decision simply on the basis of unanimity. He required that no decision be taken unless there had been prior arguments over possibilities, disagreements showing that several points of view had been considered, that not everyone was overlooking the inevitable flaws of any proposed plan.

To be able to take such historical factors into the decision-making, this information must be recorded somewhere. The details of this are still open, and there are several obvious paths to investigate. In the first method, policies are always represented as plans, and the temporal orderings on the execution of the intentions in these plans provides the required information. This, however, seems too baroque, and a second possibility is to analyze the set of reasons to tell if good arguments have occurred. A third, even simpler possibility is to just record the sets PNC, PC, CNP, and NPNC in each second-order deliberation record. This summarized information can then be consulted by examining previous reflections to see if options moved from one classification to another. By using these reflection records, POLICY-1 and POLICY-2 above might be replaced as follows. Here the predicate DEFENDED means that the option in question is now in PNC (CNP) but at some past time was either PC or CNP (PNC).

```
POLICY-3: If there is exactly one option in PNC and it is DEFENDED,
        then take it as the first-order outcome.

POLICY-4: If PNC contains more than one option, and at least one of these
        is DEFENDED, then pick a defended option randomly as the
        first-order outcome.

POLICY-5: If no options are yet DEFENDED, then do not make a decision.
```

This notion of DEFENDED might be used in another similar policy for cases in which all options seem bad.

```
POLICY-6: If all options are in CNP and are defended, then reject the decision.
```

In addition to this general rejection policy, I expect each domain would incorporate reformulation policies which would suggest specific reformulations of the decision intention or replacement of options if all options are in CNP or PC respectively. These more specific policies should override the general one.

Of course, this notion of DEFENDED is too weak. What really seems desired here is a refinement of DEFENDED which incorporates some restriction on the completeness of the set of considerations with respect to the relevancy procedures and resource limitations. The techniques discussed in [Moore 1979] may be useful in these investigating such refinements.

In general, one should consider all possibilities when making a decision. Hence the following two policies for continuing deliberation.

```
POLICY-7: If there is an unrealized consideration,
          then carry out the oldest one as the default.

POLICY-8: If there is an unrealized, non-consideration subordinate,
          then carry out the oldest one as the default.

POLICY-9: Prefer defaults created by POLICY-7 to those created by POLICY-8.
```

In some cases, policies will construct inconsistent preferences among the options. Further policies must be supplied to guide the revision of these inconsistencies. For example, POLICY-9 above rectifies the initially inconsistent policies 7 and 8, both of which declare some option to be the lowest in the partial order. However, their inconsistency would not be very serious, for RMS would just accept as the default whichever came first. But in more complicated cases (involving odd-length cycles), such as each of O1, O2, and O3 having a good reason for them, to which the policies Prefer O1 to O2, Prefer O2 to O3, and Prefer O3 to O1 are added. RMS would discover an apparently unsatisfiable circularity, and create an intention to revise these inconsistent reasons, that is, to defeat one of the preferences involved. Thus in cases like this, additional conflict-resolution policies must be supplied.

In many cases, however, there will not be enough information to argue about the options to produce a defended option. In other cases, there may be no policies which will resolve conflicts, so that to the best abilities of the program, the best options are those in PC. These are irreconcilable dilemmas for the program, and to act it might have policies like the following.[70]

```
POLICY-9: If there is pressure to decide, and all information has been considered,
        and there are still no PNC options but there are some PC options,
        then pick one of the PC options randomly.

POLICY-10: If there is pressure to decide, and all information has been considered,
        and there are still no PNC or PC options but there are some NPNC options,
        then pick one of the NPNC options randomly.

POLICY-11: If there is pressure to decide, and all information has been considered,
        and there are still no PNC, PC, or NPNC options but there are some CNP options,
        then pick one of the CNP options randomly or reject the decision.
```

It is difficult to say much more about these sorts of policies in the abstract, since most policies of these kinds are likely to be domain specific. Much experimentation and experience is necessary here.

This concludes the digression on second-order policies, and we continue with the steps of the second-order deliberation procedure.

### 5.7.3.3 Second-order Decisions

3. *Choose the second-order outcome:* The next step of the deliberation procedure, after retrieving and applying all the second-order policies, is to choose some second-order option as the second-order outcome. This choice is made by selecting the first second-order option that is in PNC in the order of preference of options D, E, C, A, B, that is pursue a consideration, pursue a subordinate, decide on an outcome, delay, and reject. It would be elegant to develop some way of making this third-level decision uniform with the second-order decision, perhaps by termination policies which always decided unless the second-order policies conflicted. There are many subtleties here, such as the fact that the third-order options are basically the same as the second-order options, that make this an

[70] These policies all act on a paucity of information, similar to NASL's QUIESCENCE choice rules.

intriguing question for further study.

4. *Act on the second-order outcome:* If the outcome is to pursue a consideration (D), this means returning to Step 4. If it is to pursue a subordinate (E), this means to add a deliberation continuation intention along with ordering policies making the selected subordinate the only intention on the frontier. If it is to delay (A), then add a deliberation continuation intention with ordering policies preferring current frontier intentions to it. If it is to reject (B), then defeat the decision intention. If the outcome is to act on a first-level option (C), then an execution procedure is retrieved for doing this, as different sorts of decisions involve different actions. For example, if the decision is about whether to form intentions from desires, then if some desires are chosen, new intentions are constructed with the aims of the desires, and added to the set of intentions. If the decision is to pick some intention to work on next, it is handed over to the interpreter for carrying out. If the decision is about some aspect of a current intention, the chosen value is inferred in that intention theory.

## 5.8 An Example Reworked

In the beginning, Robbie's interpreter is carrying out the currently active intention of passing through a door. Robbie has reached the door and is considering how to proceed, the next step of his plan being to open the door. At this point, Robbie's visual system detects an object moving towards him, and identifies the object as a woman. Robbie has a policy of normally interrupting whatever he is doing to consider what to do about approaching objects, since such object are often important to survival, either as food or as dangers. This policy suggests that he decide what to do about the woman, and defeats his first thought to continue what he was doing, namely to proceed with the next step of his previous plan and open the door.

So Robbie decides to consider what to do about the woman rather than to open the door. He begins work on the following plan.

```
(INTENTION I-1 () (AIM) (CHOOSE (ASPECT=AIM) (INTENTION=I-2) () (OUTCOME)
                                   [OBJECT AIM] = VISUAL-OBJECT-DESCRIPTION))
(INTENTION I-2 () () AIM)
(ANTECEDES I-1 I-2)
```

Here the aim of I-1 means to decide what to do about the approaching object. It takes in the object description as passed in from the visual system and outputs an aim for I-2.

The interpreter begins work on I-1 which it carries out by a deliberation procedure DP based on the above. The first thing DP does is to create a deliberation record DR. DP declares that I-1 is the purpose of DR. It then tries to retrieve the set of policies relevant to the purpose and current state of affairs. This means that the database retrieval procedures take as arguments I-1 (the purpose), DR (the current state of the deliberation), and ME (the current state of the program in general).

The first thing retrieved is the policy "A gentleman always holds the door for a lady." DP adds a consideration for this policy to the list of considerations of DR. More formally, this policy is as follows.

```
POLICY-1:  If A: the aim of the purpose of DR is to choose an aim
                        and the object of the aim is a lady-like-appearing female,
                        and there is a current intention with active progress status
                        and the aim of that intention is to open a door,
                  then (PRO (OPTION "hold door for OBJECT") (SL (POLICY-1 A) ()))
```

Here we have taken the liberty of writing an English description of the condition and the option. Actually, the condition is a logical statement of just what is said, in terms of the descriptions involved and their parts.

Following a brief reflection which decides to continue deliberation (since nothing has been done yet), DP applies this policy by evaluating its condition to see that it holds, and then executes the actions in the consequent of the policy. The first action adds an option O-1 to the (currently empty) list of options of DR, the option of holding the door open for the woman. The second action says that POLICY-1 and the application condition A form a reason for O-1, and adds this reason, R-1, to the list of reasons of DR.

DP then re-interrogates the database to see if any new considerations can be found relevant to the new items. In this case, the new option does not lead to any new considerations, but the new reason does. Since that time long ago when Robbie was initially programmed, chivalrous reasons for actions have become socially unacceptable. Robbie has learned to watch out for temptations to act chivalrously. He does this by means of the policy POLICY-2.

```
POLICY-2: If A: R is a reason in the deliberation record of the current decision
                   and R's reason involves POLICY-1,
               then (CON R (SL (POLICY-2 A) ())).
```

The condition of this policy holds, so DP executes the action, which adds a reason R-2 to DR, a reason against R-1. This invalidates R-1, so now O-1 has no good reason. DP sees it is without a good option in reflection, continues to scan the database, and finds a third relevant policy. After further reflection it applies this policy, which also has a true condition.

```
POLICY-3: If A: the aim of the purpose of DR is to choose an aim
                   and the object of the aim is a non-threatening person
                   and there is an active intention with active progress status
                   and the aim of that intention is to open the door)
               then (PRO (OPTION "hold the door for OBJECT") (SL (POLICY-3 A)))
```

Executing this policy's first action adds another reason for O-1 being an option, and the second action adds a new reason, R-3, for taking O-1. DP now finds no more policies, and again enters second-order deliberation. RMS shows that of the three reasons in DR, R-2 is valid, so R-1 is invalid, and R-3 is valid. Thus, all things considered, O-1 has a valid pro reason, so DP takes it as the outcome of the deliberation. Intention I-2 thus gets an aim to hold the door for the woman, which the interpreter then carries out, so Robbie holds the door for the woman.

# CHAPTER 6

## DELIBERATE CHANGES OF MENTAL LIFE

To survive, we must change ourselves as well as the world around us.[71] We must reflect on our beliefs, concepts, desires, values, and skills to judge whether our life would be better if we held or employed different ones.

These changes in ourselves take many forms, and are brought about for many reasons, such as to become happier, more competent, informed, efficient, to conform with others, or to become free of confusion, contradiction, or doubt. We sometimes decide to change to improve the correspondence of our attitudes with the world, or with our standards for ourselves. For example, I change my belief that a door is open because my unsuccessful attempt to walk through it points up a mismatch between my beliefs and reality. Either I hallucinated the attempted passage through the door and the pain in my nose, or I am wrong about the door's being open. Or as another example, I wish to become a mathematician, only to find that my intuitions conflict, that I believe that the irrationals far outnumber the rationals, but infer a conflicting belief from the existence of an irrational between each pair of rationals and a rational between each pair of irrationals. In this case I cannot give up either of these beliefs, as they are part of what mathematicians believe, so I must give up my inference that they conflict. Or finally, I judge my inference that I am a terrible person because I can't sing well to be the cause of my unhappiness, and thus of the mismatch between my observed unhappy mental state and my standards of a happy outlook. To remedy this mismatch, I either give up the inference that I am a terrible person, or the desire that I be happy. But these changes do not just happen. In most cases, it is my realization of the need for change which leads me to decide to change, to form an intention to change, and then to carry out that intention.

[71] What is survival? If we are mutable, what is it that is surviving? Throughout this thesis we maintain the fiction that there is something called the "self." Chapter 2 presented some general reasons why this is desirable, but this thesis is not the place for the discussion this question deserves. I hope to analyze this question in light of the current model in a later paper.

Deliberate adaptations perhaps play a larger role in developmental psychology than is normally recognized. For example, many accounts of the psychological development of children are puzzled by the apparent inexplicability of the changes undergone by the child. The answers to these puzzles may often be that the child at some point realizes that he is frustrated by an inability to perform some task, and simply decides to learn how to do it. Such deliberate changes are more clearly recognizable in the case of adults who, for instance, decide to take classes to acquire some skill or knowledge.

This chapter describes how to use the techniques previously developed in this thesis to deliberately change the content of the program's mental life.[72] In all cases, the basic recipe for change is similar. The motives for changes come through reflection, and the implementation of changes comes through intentions to change. The program first reflects on its set of attitudes, by using its self-referential ability to view its current set of beliefs, desires, skills, concepts, or values, and to infer properties of that set which indicate the desirability of change. The reflection occurs during deliberation on what to do, and policies recognize the motivating conditions for changes. This reflection is followed by further deliberation and planning of what changes might be appropriate and which changes should be taken. Further policies guide this decision of how to change, and the result is an intention or plan for implementing the change.

For example, a policy applied during reflection may reveal an inconsistency in beliefs, or an unexpected, erroneous effect of an action. The program may then take these realizations of contradictions or bugs in procedures as cues to correct itself, and form intentions to fix the incorrect assumptions or procedures. The program can then apply itself to deliberately tracking down which beliefs or procedures are at fault. These changes might be carried out by simple techniques, such as

[72] A large problem, if it can be called a problem for a reasoner rather than for the geneticists and psychologists of a species, is how to change the form of one's mental life, how to choose and invent or discard various emotions, e.g. creating an intelligence that lacks fear, or combativeness, or other attributes. These are rarely problems for the individual (except perhaps in Buddhists), as he is more frequently concerned with questions of how to improve his knowledge of the world, how to stop being depressed, how to enjoy life more, how to stop smoking, how to perform his job better, etc. It is these more circumscribed changes that we deal with here.

dependency-directed backtracking [Doyle 1979], automated debugging techniques [Sussman 1975], or even asking the user for help. Its plans for carrying out these changes might be very involved. Faults in primitive procedures can take much experimentation, simulation, and analysis to locate (as any programmer can tell), and false beliefs can require similar searching out (as psychiatrists will vouch).

By and large, these techniques of deliberate changes are familiar to AI, as they are the sorts of imperative changes programs make on their own data-structures. In most AI programs, imperative operations are used from the start and taken for granted, because most programming languages are founded on imperatives. In contrast, imperative changes come near the end of this thesis as applications because we concentrate on the reasons for these changes, which normal imperative languages ignore. When one sets a variable in LISP, one rarely can tell why that change occurred. That is part of what makes debugging programs so hard. What we aim for is ways of performing the same operations, but so as to be able to explain and analyze them later.

The reader is cautioned that the rest of this chapter is exceedingly vague, more in the way of hints for future research than presentation of concrete techniques. Unfortunately, time limitations have precluded presentation of anything but a sketch of motivations and methods for change. Most of these sketches merely refer to other works where these sorts of changes have been studied in their own right. Casual readers are encouraged to skip to the next chapter, as the basic ideas of this chapter have been presented in this prologue. The remainder of the chapter contains only slightly more concrete examples.

### 6.1 Motivations for Change

In this section we catalog a variety of the policies which might be employed during reflection to lead to intentions to change the program's attitudes. Each of the policies we describe is of the form "If the current set of attitudes has property P, then reason for the option of making change C." Of course, during deliberation, the sets of attitudes are changing constantly, so the set S of attitudes whose properties are inferred in the condition of the policy will usually not be the set of attitudes after the policy has been applied. However, we, and the policies we write, ignore this problem and (except for special kinds of policies mentioned later) always assume that the properties in question are invariant under deliberation. This is usually a safe assumption for properties like "is inconsistent" or "contains no procedure for installing light bulbs" are rarely affected by deliberation alone.

### 6.1.1 Belief

The major reasons for changing one's belief are to explain some unexpected fact, to cope with surprises while taking actions, to resolve conflicts, and to adopt or abandon beliefs with specific long-term consequences in actions or otherwise. Properly, the following policies describe changes to the set of inferences recorded as justifications, since the program derives its current set of beliefs from the current set of justifications.

*B1:* If someone informs me of a fact, try to explain it from my previous beliefs, or try to detect its inconsistency with them.

In general, one always seeks to explain surprising facts, but as far as I know, no completely adequate account has been given of what surprising beliefs are, why one wants to explain them, or exactly what it means to explain them. Schank [1979] classifies new information by subject matter and uses these classifications in deciding whether or not to investigate its consequences.

*B2:* If the observed effects of an action conflict with the effects I predicted, then try to explain the failure of the predictions.

Observations might lead one to abandon conflicting predictions, but they rarely explain the failure without further explanation.

*B3:* If at some times I seem to act as though I believed B and at other times as though I believed ¬B, try to determine which I believe and make me do so consistently.

Often one reflects on one's actions to justify or rationalize them to oneself. This is particularly true of actions carried out unconsciously (as in primitives). These rationalizations involve constructing imaginary desires, beliefs, and intentions which would have lead to the action, that is, pretending the action had been taken to carry out an intention directly, and asking what that intention was and why it was held. If this process gives seemingly incompatible rationalizations on different occasions, there may be some confusion which can be clarified.

*B4:* If the current set of beliefs is inconsistent, then try to remove the inconsistency.

Here the set of beliefs is inconsistent if it contains two beliefs A and B such that A∧B is contradictory.

*B5:* If the current set of justifications contains an unsatisfiable circularity, try to make it satisfiable.

This is no. a condition ordinarily recognized during reflection, but rather a condition noticed by RMS. These unsatisfiable circularities can be viewed as describing paradoxical statements or inferences that cannot be taken as either true or false, or valid or invalid. The simplest response to this condition is to reject the final inference to paradox, to ignore it, as when one laughs upon being told Russell's paradox.

*B6:* If the current set of non-monotonic assumptions about things currently supports an unhappy, depressed, frustrated, or other undesirable outlook, and the same set of non-monotonic inferences can support by reinterpretation a happy or other desirable outlook, then try to switch the interpretation of these assumptions to the happy or more desirable outlook.

This policy expresses a policy similar to B4 about inconsistent beliefs. There are many reasons one might avoid certain patterns of beliefs, not just that they are inconsistent, but also that they have other bad qualities besides the confusion caused by inconsistency. The message of many self-help books is that while sometimes our unhappiness results from pain and other true discomfitures, frequently our unhappiness is merely an interpretation we needlessly impose on our beliefs, that is, merely a set of inferences better left unmade. For example, one might feel bad because one makes the inference "I'm a terrible person because I'm an incompetent singer." The solution is to recognize oneself making this inference and avoid it, in the same way one might avoid taking the final step of the argument to Russell's paradox. One avoids making the undesirable inference and cultivates instead alternate inferences from the data, such as "I should take voice lessons because I'm an incompetent singer," or "It's good I enjoy singing for myself, because my incompetence would really aggravate others," or " I can earn tidy sums by singing until people pay me to stop or leave."

*B7:* If the current set of beliefs contains beliefs which might have undesirable effects in the future, then try to change to beliefs which do not lead to undesirable effects.

Where B6 notices currently annoying aspects of beliefs, B7 attempts to anticipate possible future annoyances. A contemporary example of such a change is the business executive who becomes a Republican to avoid hindering future promotions made by Republican superiors. The classical example of such a change of belief is Pascal's wager. Pascal believed that if God exists, then He must have the traits attributed to Him by the Christian Bible. Pascal reasoned that if he had faith in God, then at worst he would miss out on life's voluptuary pleasures, and at best he would gain admission to Heaven, which for him was by far the most one could hope for in any mode of existence. He reasoned further that if he withheld faith in God, at best he would sample life's voluptuary pleasures, and at worst would suffer infinite torment in Hell. Pascal judged the eternal possibilities more important to him than the transitory human opportunities, and adopted the Christian faith.[73]

[73] His musing on this question was the cause of his faith, but not its reason. That is, his deliberation lead him to form an intention to adopt this faith. The intention depended on the prior beliefs. The faith did not depend on the prior beliefs, for it was purely an effect of an action taken to carry out the intention. While the intention is the cause of the action taken to satisfy it, the action record on which the faith depends is an observation, a premise, of the program about itself, and does not depend via reasons on the intention. There are many interesting subtleties about the nature of action here, but we will not pursue them now.

### 6.1.2 Concepts

Since concepts or the theories in the hierarchical database do not refer to the world, but rather are used by attitudes in referring to the world, it does not make sense to speak of a concept as an attitude, of a concept being incorrect because it does not match reality. If it did, we would have to conclude Pegasus to be an incorrect concept. Rather, the following policies create and revise concepts on the basis of completeness, efficiency, and correctness with respect to a shared vocabulary among discussants. Since there is a large literature on concept formation and revision, which suggests many policies for these changes, I merely present a few of the most basic ones.[74]

*C1:* If the same combination of concepts (e.g. a log from one ground to another) is constructed on two occasions for different problems (traversing a stream and a crevasse), create a new concept (bridge) whose structure is that combination.

*C2:* If one concept (e.g animal) has too many specializations (dog, perch, horned toad) in the hierarchy for efficient searching, create new intermediate concepts (mammal, fish, reptile) to decrease the branching factor and capture commonalities.

*C3:* If people persistently seem to misunderstand one's use of a concept (e.g. elephant), investigate their concept to see whether they mean the same thing (that large African quadruped with the round face, big teeth, that spends a lot of time in the river and swims under and upsets boats).

[74] See Winston [1975], Fahlman [1979], and Fox [1978].

### 6.1.3 Desires and Intentions

Like incorrect beliefs, unsatisfiable desires can sometimes lead to injury or frustration, so care must be exercised in deciding which desires and habits one inculcates or breaks. Intentions are usually more transitory than one's basic desires, but without frequent review of one's plans it is easy to fall into continuing to carry out intentions whose reasons have long since departed.

*D1:* If a desire for the forseeable future leads only to undesirable effects, such as frustration through one's inability to satisfy it, and to no redeeming influences on one's actions, then attempt to abandon the desire.

For example, I might abandon my desire for drinking soft drinks, as they are often without redeeming feature and not without unsavory aspects, but I might not abandon my overindulgence in book-buying, as there are almost always good aspects of this problem.

*D2:* If a possible desire might have desirable influences on one's behavior, try to inculcate it.

Many people, for example, develop a desire for regular exercise to improve their vigor.

*D3:* If someone admired expresses certain desires and not others, try to emulate that person by inculcating a similar set of desires.

This sort of policy is often part of a large plan when the admired person is a potential friend, as when one adopts new interests so as to be able to converse at length with someone.

*I1:* If one holds an intention because it is part of a plan, the justification (or superior) of which has been defeated (abandoned), and the intention is not necessary for cleaning up after previously executed intentions, then abandon the intention.

Of course, to this short list should be added the many planning techniques which rely on reflecting on one's intentions, such as those of Sacerdoti [1977] and Tate [1975]. These policies include resolving inconsistencies in one's desires and intentions.

*I2:* If one holds inconsistent intentions (e.g circular priorities between intentions), change them to restore consistency.

### 6.1.4 Values

The program's values as embedded in policies can be reflected on to increase their coherence and completeness.

*V1:* If one's values have, during deliberation, proven to be inconsistent or paradoxical, then try to modify them to avoid similar problems in the future.

Here policies are called inconsistent if they draw opposite conclusions from the same data, such as "If it's raining, then go inside" and "If it's raining, then stay outside". Policies are paradoxical in some cases if their application leads to preferences with multiple interpretations or unsatisfiable circularities in RMS. These paradoxes result from the fragmentation of value, from the need to make unitary decisions based on disparate considerations. The paradoxes manifest themselves most familiarly in non-transitive preferences between options, which make the result of deliberation depend not only on the reasons for and against the options, but also on the order in which they are considered. The typical example of such a situation is in, say, political campaigns, in which one prefers candidate A to B, and B to C, but prefers C to A, and so prefers A if they are presented in the order CBA, but prefers C if they are presented in the order BAC.

*V2:* If one's values have, in many deliberations, proven to have consistent results after much reasoning, then summarize the net decisions in new policies which are based on but replace in action the previous policies.

Sometimes I find myself going through the same old arguments each time the same decision confronts me. In these cases I often step back and decide the question once and for all (barring irresoluteness or later information being discovered). For example, I never rehash the arguments for and against holding doors for people, as I decided long ago to always hold doors, and to handle problems with this approach as they (infrequently) arise.

*V3:* If one is frequently confronted with a dilemma which is always broken randomly, adopt some new value to avoid the effort of this decision.

### 6.1.5 Skills

As in the case of belief, there are many sorts of reasons for modifying one's set of skills, which we will interpret to mean one's procedures, both plans and primitives, along with their method statements.[75] Changes to the set of skills include both developing new skills and modifying existing skills, there being a number of reasons for modifying skills.

The basic case of skill development is that of one-time construction in problem solving, when one puts together a plan for solving a problem which may or may not be retained in the library of procedures. New skills are constructed from old ones, either by combining several procedures in some arrangement, or by modifying a copy of a procedure for some similar problem.

*S1:* If one will need in the future to achieve some aim by some means satisfying some specifications, the construct such a procedure, index it under that aim, and describe it with those specifications.

The specification of procedures, as we have touched on previously, is still an active area of study, as these specifications can refer not only to input-output behavior, but also to complexity, explicability, intermediate states, and other aspects of the process.

An important part of one's skills is the description of the procedures. These descriptions serve

[75] Policies are parts of plans for deliberating, and the previous subsection mentioned how one might make deliberating more efficient by reorganizing one's set of policies.

not only to index the procedures so that they may be considered when relevant, but also to specify their intended and observed effects. A common cause for modification or maintenance of a skill is when a mismatch develops between these descriptions and the reality of the procedure's capabilities. These mismatches can result from changes in the program's attitudes, changes in the patterns of use of the procedure, changes in the physical realization of the program, or changes in the physical environment of its realization. For example, I must modify my speaking skills when I find myself committed to teaching my first class. I must modify my motor skills as I grow older and the physical realizations of my procedures fails to match what I think they can do. I similarly must modify my motor skills if I move to Luna, where my previous skills no longer have the intended effects. Other, less general mismatches occur when applications of the procedures in novel circumstances discovers failures or other unexpected results.

*S2:* If a skill fails to achieve its expected effects in a normal situation, then it is broken, so modify it to restore its functionality.

*S3:* If a skill fails to achieve its expected effects in an exceptional or unconsidered situation, modify the set of skills to cover this case as well.

*S4:* If a skill achieves its expected effects but has undesirable side-effects, repair it to avoid those side-effects.

*S5:* If a skill has unexpected but desirable effects (serendipitous performance), analyze it to extract a skill for these desirable effects.

## 6.2 Mechanisms of Change

As we sketched previously, the mechanisms for these changes are procedures in the library of procedures. The techniques employed are based on an analysis of the reasons underlying the attitudes to be changed, since to be an effective change the program must modify not only the attitudes directly under consideration, but also those underlying them in their reasons.

### 6.2.1 Belief

The basic approach towards belief revision suggested here is that of incremental revisions guided by policies expressing preferences between alternate partial revisions. In other terms, the policies express the relative tenacities with which the program holds its beliefs. This means that the program begins revising its beliefs by deciding on some particular beliefs to change. As it attempts to change those beliefs, it discovers that further decisions must be made about how to accommodate the changes in the remaining beliefs. These steps of decision and partial revision alternate until the system of beliefs has been coherently modified in accordance with the intended revision.

This sort of revision accounts for the policies B2, B4, B5, and B6 above. B2, B4, and B6 are about changing beliefs, and B5 is about fixing the set of justifications for beliefs, but since we make all changes in beliefs by adding and defeating justifications, we can handle all of these changes using the same techniques. We view unsatisfiable circularities as inconsistent specifications for the set of beliefs, inconsistencies in the reflected justifications. Similarly, we view the undesirable conditions of B2, B4, and B6 as inconsistencies. B4 concerns inconsistencies directly. B6 we interpret as an inconsistency between actual beliefs and intended beliefs, and B2 we interpret as an inconsistency of action or predictions of those effects, where the predictions are inferences from the action record and the action specifications.

However, matters are complicated by the ambiguity of belief revisions. When beliefs derived by inferences or actions conflict with previous beliefs, there are many ways of reconciling the conflicting belief. Any participating belief may be rejected, not just the previous beliefs, and what revision is made depends on the context of the inconsistency. For example, the ways of resolving an inconsistency are different depending on whether the program was just thinking through an action (planning), or whether the program actually took the action. If the action was a hypothesized part of a plan, the program might choose to discard the action and try another. If it actually took the action, it might discard the action (and so think that the action was hallucinated) or find some assumption about the world that must be wrong. Suppose the program tries to lift a large object via a cable on a crane. If it lifts the crane and detects that the object still rests in place, it might reason that either it imagined lifting the crane, or that its senses reporting that the object remained unmoved are wrong, or that its assumption that the cable would hold the object was wrong, that it snapped.

This problem of ambiguity of belief revision leads to one of the three forms in which the interpreter makes decisions via decision intentions. If RMS reports an inconsistency following a primitive execution, or an ambiguity in the revision necessary to incorporate the primitive's effects, the program reflects on this ambiguity by creating a decision intention. In the case of an inconsistency, it is an intention to decide how to remove the inconsistency. In the case of a direct ambiguity, it is an intention to decide which of the alternatives to take.[76]

Now primitives should rarely lead to deliberation about how to revise beliefs. If they are properly organized, they will do all the necessary belief revision directly. The basic idea here is that "properly organized" means that the primitive action or revision procedure is a procedure compiled from more complex deliberation procedures by specializing the processes to take into account the usual-case information about the effects of that particular action. For example, a primitive which updates some list kept as an attached value might justify the new attachment and defeat the justification of the previous

[76] It would be a very interesting task to encode RMS largely as policies guiding deliberate changes of beliefs, so that RMS would take the form of a MACRO-RMS/MICRO-RMS combination analogous to MACRO-TORPID/MICRO-TORPID. This might be developed into a belief system closer to human belief systems than the current RMS.

attachment with a justification mentioning the new attachment.

Action-specific belief revision procedures incorporate information about how the action normally affects beliefs: what sorts of beliefs are normally involved, what the normal alternate revisions are, and which revision is the usual one, that is, which beliefs are normally rejected by the action and which new beliefs normally take their place.

This information about the normal alternatives and preferences in belief revision is stated as policies which suggest and discriminate between revisions. For example, one might tell a human "If you feel cold after taking this drug, it is because of the drug and not because it is cold outside." This policy would be very useful in explaining conflicts between a feeling that it is cold outside and observations of a thermometer and the sweltering of others indicating that it is hot.

As another example, assume that the program is being used to solve problems of manipulating a set of blocks with a one-arm manipulator. Here we might give the program information about the normal effects of the manipulation primitives. Two different such policies to guide its decision might be as follows.

(1) When a block is moved from one place to another, give up the belief that it occupies the current location rather than rejecting the conflicting belief that it now occupies the new location. (Of course, this looks much like the add list/delete lists used in STRIPS.)

(2) When planning actions rather than taking them, if a block is moved and a conflict arises between the belief about the block's new location and the location of some other block (a collision), give up the action and its effects rather than the belief about the other block, and then plan a different action (perhaps one to get rid of the obstacle block followed by the current action).

There are similarities in spirit between this formulation of action effects and some previous approaches to belief revision. As mentioned above, Strips' add and delete lists [Fikes and Nilsson 1971] were essentially policies which specified which of the several possible revisions to take. Rather than just using a modal statement of the action effect, e.g. After A, P is true and Q is false, and letting these two statements conflict with the existing database statements, the add and delete lists say, e.g. After A, take P rather than ¬P, ¬Q rather than Q.

Another technique is the use of "gripe handlers" (or "complaint departments") introduced in BUILD [Fahlman 1974]. These are procedures provided explicitly to discriminate between the revisions possible following the discovery of an inconsistency. The gripe handler of the procedure taking some action might be invoked with the information that the action caused a collision (a conflict between two beliefs about block locations), or an unstable structure either at the source or at the target of the moved block, or other errors. The gripe handlers in BUILD never rejected beliefs about the blocks in question, but always rejected some action or actions in the current plan. The gripe handlers would classify the error type (collision, instability, etc.) and would either reject some action itself, or would look at the goal structure of the plan and pass the problem off to the gripe handler of some specific other action. These gripe handlers seem very similar in conception to revision procedures, save that they only reject actions in the plan rather than beliefs in general.

A final technique for comparison is that of resolution rules as developed in AIMDS [Sridharan 1976, Sridharan and Hawrusik 1977]. These are also close in spirit to our revision procedures. AIMDS splits belief revision into two sorts of rules: recognition rules, which are statements of logical and causal dependencies between the primary effects of the action and other beliefs, and resolution rules, which are rules for selecting one of the revisions possible given the related beliefs computed by the recognition rules. While it is claimed possible for AIMDS to generate the recognition rules itself (by rephrasing the logical axioms describing the domain to summarize chains of inferences), the examples presented do not contain all dependencies, and thus do not allow any belief to be rejected. Also, the system does not use the resolution rules as a way of deliberating about what change of belief to make, but interprets them as imperatives. That is, if there are a number of (possibly incompatible) resolution rules, AIMDS tries them one-by-one until the action of some rule is not rejected by the database, rather than realizing that there is a decision to be made about which resolution rule to use. Also, how the database decides to reject a proposed change is not spelled out, although this involves values implicitly.

I have nothing to suggest about how to handle policies like B3, as it deserves further study.

B7 describes a "leap of faith." This can be implemented by justifying the belief as a premise and by adopting policies to defend the belief during belief revision. Thus if the program wishes to have faith in the statement "I believe in God," it first asserts this belief as a premise. (Actually, the adopted belief depends only on the realization record for the belief-adoption action. In this way the adopted belief is recalled as having been adopted, but does not depend on other beliefs, such as those which lead to its adoption.) The program similarly can adopt policies as premises which defend the belief against change in any inconsistency, action, or other revision process. Perhaps much of the difficulty humans have in adopting new positions and making them stick stems from the relative ease of adopting a belief as opposed to adopting also all the policies and procedures necessary to make the belief enter effectively into actions and decisions.

### 6.2.2 Concepts

I will not go into techniques for revising the set of concepts at all, as this topic is adequately covered in numerous other works, as far as it has been explored. As usual, however, alternate ways of revising the set of concepts will be the subject of deliberation and policies will embody the program's values concerning organizations of its database.

### 6.2.3 Desires and Intentions

Sacerdoti [1977, 1979] explains a number of techniques for reflecting on ordering policies and other intentions in planning. Shrobe [1979b] discusses how reflection on desires and intentions allows their revision upon satisfying one particular desire or intention, using reason-analyzing techniques, but without deliberation. Basic desires and policies are much like premise beliefs, and the techniques for inculcating and abandoning them are similar to those for leaps of faith, although they normally need not require further defensive policies.

### 6.2.4 Values

The question of how to revise values and their embodiments in policies is unexplored as far as I know, and neither have I pursued it here.

### 6.2.5 Skills

HACKER [Sussman 1975] learned procedures for manipulating hypothetical blocks with a hypothetical one-armed manipulator. It started its career with a couple of primitives for the manipulator, a store of general programming and planning tricks, a few facts about the world of blocks and about its manipulator, and a store of general ways to analyze and correct bugs in programs. When presented with a problem, HACKER would either remember or construct a program for solving it. If it constructed the program, it did so either by generalizing a piece of code used for solving a similar problem in some other program, or by using general planning techniques to combine its own primitives to achieve complex conditions. If the remembered or constructed program worked, Hacker remembered it and went on to the next problem to be solved. If the program failed, however, HACKER performed a ritual self-examination to correct the program if possible. It would first construct a description of the "process" in which the error occurred, this including the history of the executed actions, their effects, their teleology, and the intentions being carried out. It would then ask several questions about this process model to determine the bug type. Some questions were counterfactuals, i.e. could such-and-such a step have been inserted without conflicting with other goals at that time? Other questions matched certain abstract process models against the actual process model to see if it realized the bug type associated with the abstract process model. The answer to these questions was the type of bug underlying the error.

HACKER then searched the library of bug-patches with this bug type and with the patch located patched the failing program. HACKER repeated the tryout and fix cycle until either the program worked or until no way could be found to solve some problem, in which case HACKER gave up.

However, the trouble HACKER went to in analyzing its bugs resulted in large part from its lack of the sorts of techniques we have developed in this thesis for representing the reasons and intentions of the program. For example, all of the information HACKER painfully sifted from Conniver contexts and control stacks in building its process models is exactly the sort explicitly available in justifications, the sets of desires and intentions, and the action history.

Since skill modification is such an important part of efficient and effective action, especially in a program whose careful operations are as complex as ours, we illustrate the ideas developed in the previous chapters by reformulating HACKER using our techniques. This reformulation also raises a number of topics for future research, particularly hypothetical reasoning and historical reconstruction, which we hint at but have not pursued in the detail they desei ve.

HACKER involves three major plans:

1. DEVELOP - for developing a new skill from scratch,
2. CRITICIZE - for patching a known bug in a program under development, and
3. DEBUG - for fixing a program manifesting an error.

We present these plans informally in English.

## DEVELOP

1. *If the skill is in the procedure library, DEBUG.*

This step retrieves a procedure for an intention via the usual method statement techniques used by the interpreter. DEBUG will carefully test the procedure to see if it works, and patch it if it does not.

2. *Otherwise, construct a new procedure.*

HACKER uses two methods to construct new procedures.

The first method is to generalize or variabilize part of a plan used to solve some previously encountered similar problem, and make this a new plan. At the same time, this part of the plan is replaced in the plan it was extracted from by a call to the new plan with the appropriate arguments. In this way, any improvements made to the new plan are automatically shared by the original plan.

The second procedure construction method is to apply general problem solving techniques of problem reduction, etc. to come up with a new plan by combining other plans. We won't go into this familiar subject.

More learning occurs when the first of these techniques is used, for in it many procedures are simultaneously improved and extended. The second method is more difficult than the first. Not only are the general problem solving techniques quite expensive, but in addition debugging a new program is more difficult, since several bugs may be introduced at the same time, thus making bug localization and analysis very complex.

3. Perform *CRITICIZE.*

4. Perform *DEBUG.*

5. *Compile the working program.* Just as programs in ordinary programming languages can be compiled into machine code, plans can be compiled into more specialized plans and into primitives. The basic idea is just to take a plan and some restricting conditions, such as expected initial circumstances, or a particular library of procedures and policies, and then to symbolically execute the plan under these restrictions and make a more specialized plan or primitive from the decisions made and actions taken in the symbolic execution. Plan compilation involves all the techniques standard in ordinary compilation, such as constant folding, dead code elimination, loop optimizations, etc. In addition, the plan compiler uses policies about when to coerce independent steps of a plan into a sequence, when to replace deliberations by conditionals computing the outcome of the deliberation, when to substitute subplans or primitives into plan steps, and when to transform information passed through plan variables into information stored in local data-structures.

CRITICIZE

1. *If there are criticisms of the program, patch it.* The program critics of HACKER and the plan critics of NOAH had essentially the same form, that of looking for occurrences of subplans and replacing the faulty subplan with a new one. For example, HACKER would look for steps in the wrong order and reverse them, while NOAH would look for improperly unordered steps and order them. We phrase these sorts of criticisms as policies. Thus this step consists of a decision intention to formulate and choose between possible revisions of the program. To avoid incompatible changes, only one revision is selected, and the plan recurses to effect further necessary modifications.

2. *If it was modified, CRITICIZE.*

DEBUG

1. *If it works, done.* A proposed program is tested to see if it works not by direct execution, which would leave no information to analyze an error with, but instead by symbolic execution. In symbolic execution the temporal situations occurring before and after each program step are modeled as theories copying the current state of mind. The initial conditions are stated in the initial situation, and the actions are simulated by applying their specifications or descriptions. This involves, for example, taking a Floyd-Hoare specification $P \supset [a]Q$, trying to infer P in the prior situation, and if successful, concluding Q in the subsequent situation. All specifications of each action are so applied, and a directed acyclic graph of situations results.[77] The symbolic execution halts either when the simulation is complete or when an inconsistency or other problem is inferred in one of these situations.

It would be more attractive to simply use the interpreter to carry out this simulation directly, without recording explicit temporal situations. However, this would then necessitate the ability to reconstruct past situations from finished intentions and the action history. As Chapter 7 explains, this is a

[77] Shrobe [1979a] explains this technique in detail.

difficult problem awaiting solution.

2. *Classify the bug.*

This procedure analyzes the reason for the error by asking questions about the structure of reasons and actions leading to the error. The goal is to take the surface manifestation of the error and reconstruct the underlying bug type. This is done by asking certain hypothetical questions about the surface manifestation and by matching the surface manifestation against a variety of abstract "process models" to determine the appropriate classification of the bug type.

There are four basic types of surface manifestations of errors: *unsatisfied prerequisites*, in which some condition necessary for the application of some primitive did not hold at the appropriate time; *protection violations*, in which one action interferes with conditions protected by some other ongoing action; *failed actions*, a catch-all category which ought to be refined, intended to include mechanical breakdowns, slippages, overlimits, hardware errors, etc.; and *deja vu*, my version of HACKER's double move "error." This is really not an error as such, but humans seem to be very good at recognizing certain types of repeated or similar situations, and get a lot of mileage out of recognizing them. This is generalized to any noticed similar repetition, from HACKER's which only caught repeated movements of the same block.

There are five basic underlying bug types: *prerequisite clobbers brother* (PCB), in which achieving one prerequisite of some action undoes the previous achievement of some other prerequisite of that action; *prerequisite missing* (PM), in which the plan lacks actions to achieve some condition prerequisite for taking some action; *prerequisite clobbers brother goal* (PCBG), in which achieving a prerequisite of one action undoes the effect of some other action which together with the first action worked to achieve some complex end; *strategy clobbers brother* (SCB), in which performing one strategy uncovers new information which might allow a previously failed strategy to succeed; and *anomalous*, a catch-all bug type for those errors unclassifiable as any of the preceding, which should be refined into useful categories.

Sussman presents the flowchart shown in Figure 13 for performing the classification of surface manifestations into bug types. The decisions are as follows:

1. Would U.-P. being true now conflict with the current goals?
2. Was the U.-P. ever true before in this problem?
3. Pattern-match to see if PCBG.
4. Pattern-match to see if SCB.
5. Several pattern-matches to see if PCB.
6. Pattern-match to see if PCB or PM.

These questions are answered by much the same techniques as used in HACKER, and I won't go into the details of just what sorts of patterns the various policies recognize.

3. *If it is memorable, summarize the bug.*

One should not bother remembering dismissed errors or trivial mistakes like fingers slipping while dialing a telephone number. In this step, the program deliberates on whether to record the bug as a policy which will recognize and patch its future occurrences in new programs. This involves trying to explain the error as a one-time affair, or a something that is likely to recur. As far as I know, no one has explored grounds for making these decisions.

4. *Patch the bug.*

This step just applies the selected critic policy to the plan being criticized.

5. Perform *CRITICIZE.*

6. Perform *DEBUG.*

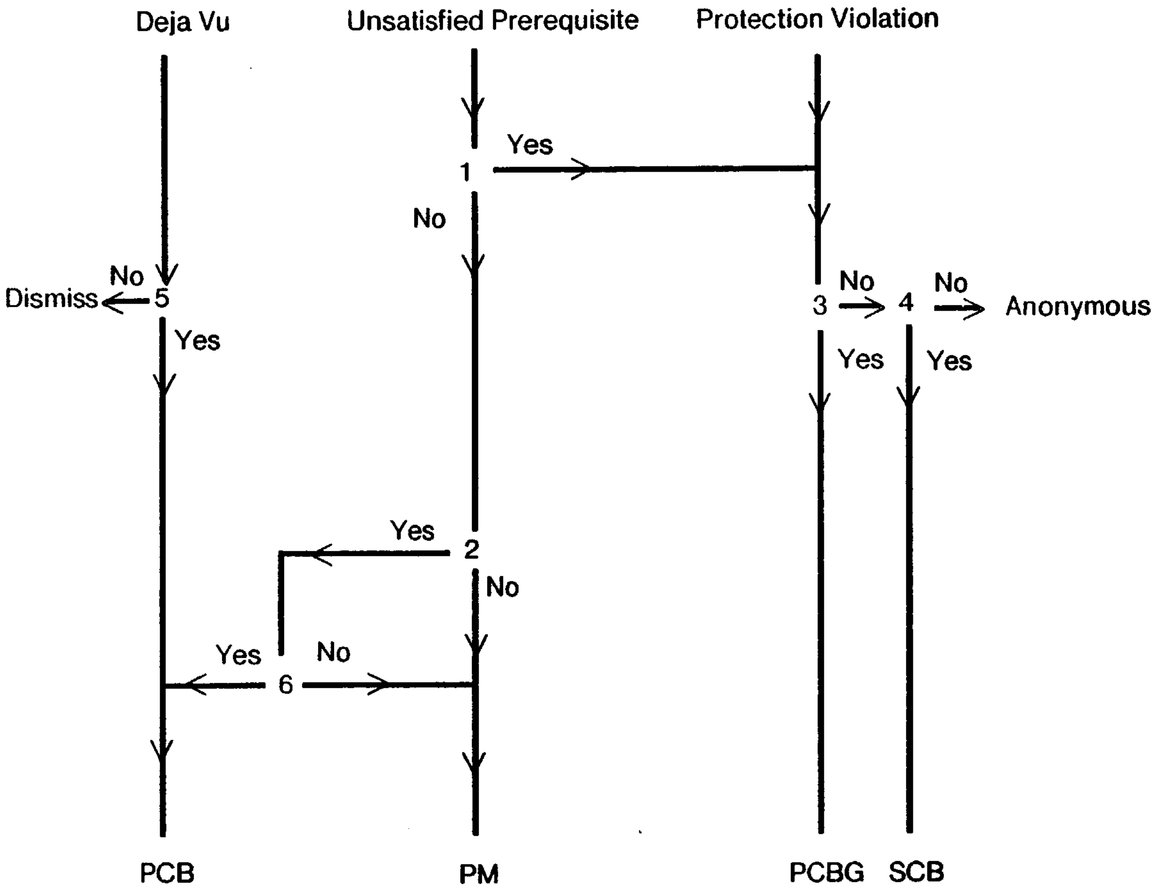


Figure 13

HACKER's Debugging Flowchart

# CHAPTER 7

# DISCUSSION

If it is not true, it is a happy invention.
Anonymous, 16th century

In this thesis, I have attempted to present some problems and viewpoints I feel are central to the task of designing intelligences. I will be satisfied if the preceding has succeeded in conveying the nature and importance of these problems and viewpoints. The techniques presented here are admittedly rudimentary and ill-explored, but they have been developed sufficiently to indicate the feasibility of this approach. However, none of the details of any technique herein is suggested as the final word; they all deserve to be superseded by more careful analyses, further experimentation, and application.

This chapter is in six parts. The first two parts summarize the key ideas and the principal technical contributions of the thesis. The third section lists a number of directions for future research. The chapter closes with three rather speculative sections concerned with the relation of affect and intellect, the limits of self-knowledge as seen in this approach, and the meaning of the program to itself.

## 7.1 Summary of the Key Ideas

The primary idea of the thesis is that of an architecture for a reasoner which can refer to, reason about, and modify any aspect of its own organization and behavior. This *self-conscious, adaptive architecture* is motivated by the need to carefully consider what to do when solving difficult problems and when carrying out complex tasks. The self-referential abilities of the reasoner are based on a meta-theoretical database, explicit reasons for attitudes, and explicit sets of the reasoner's beliefs, desires, intentions, and skills. The *meta-theoretical database* allows both self-reference in the large (the reasoner referring to itself as a whole) and self-reference in the small (the reasoner referring to its parts). Self-reference in the small allows the program to treat its own concepts and descriptions as objects. This permits not only treatment of a number of classical problems in representation theory, but also allows the efficient organization of the database into a hierarchy of concepts. Explicit, *non-monotonic reasons* form the basis of the reasoner's self-representation of its reasoning actions. These are used in *defeasible reasons* in a form of decision-making called *reasoned deliberation*, which reflects on these reasons to conduct *dialectical argumentation* about the possible outcomes of the decision. Non-monotonic reasons also form the basis of the reasoner's self-explanatory and self-modifying abilities. The *explicit sets of attitudes* form the basis of the reasoner's actions. The program reflects on itself and its current state of mind as captured in its current sets of attitudes to take actions including revising of the sets of beliefs to remove an inconsistency, forming an intention to pursue a desire, or carrying out an intention by means of some procedure (either a *plan* or a *primitive*) in the *hierarchical procedure library*. This procedure library contains part of the self-description of the program in the form of *meta-circular interpreters*, giving the reasoner a representation of its own procedures in its own language of problems and actions. Unlike many traditional studies in AI, we separate the notions of goal into *desires* and *intentions*, to make clearer the processes involved in complex problem solving reasoning and actions. Certain intentions, called *policies*, act as intentions to reason in certain ways during deliberations, and so embody the *values* of the program.

## 7.2 Summary of the Principal Contributions

The main contribution of this thesis, I feel, is in a coherent, if incomplete, synthesis of a number of important ideas developed by a number of authors. I hope that this synthesis points up directions for future investigation, and that it helps articulate some of the ideas I believe have been held by the authors I draw from. In addition to the synthesis of many important ideas, the thesis has presented novel technical contributions on the following topics, in order of their appearance.

Chapter 2 presented the basis of the correct interpretation of virtual copies of descriptions in logical terms, namely as substitution and inference of meta-theoretical statements. This was used in the construction of propositional attitudes, and in the correct interpretation of "context" mechanisms, wherein concepts and beliefs augment the current set of concepts and attitudes.

Chapter 3 presented uniformly defeasible reasons, the correct basis for adaptive and reflective reasoning programs.

Chapter 4 emphasized the advantages of desires and intentions over ambiguous "goals," the interpretation of policies as intentions to reason in certain ways during deliberations, and the correct interpretation of of procedures as partial states of mind which in execution augment the current state of mind. We also presented a meta-circular reasoning program.

Chapter 5 introduced reasoned deliberation, the first class of formal decision-making procedures to correctly account for reasons, dialectical debates, reflection, and the fragmentation of values.

Chapter 6 introduced deliberate changes of the mental state and their importance in explainable and correctable self-modifications.

The last part of Chapter 7 will draw a new conclusion about the paradox of human existence.

## 7.3 Directions for Future Research

As mentioned earlier, almost every concrete technique developed here should be viewed with suspicion of shortcomings. The preceding chapters have on occasion mentioned some of these shortcomings, and this section catalogues some of the incompletenesses not mentioned in detail previously. These topics deserve further study, and in some cases are crucial to the construction of a fully operative program, but I have not had the time or inclination to pursue all of them in this thesis. I am convinced that none of these holes harbors a homunculus, but that is something only experimentation can demonstrate.

1. *Make virtual copies virtual:* SDL, as implemented, actually copies all its copy theories, resulting in a real pile of data-structures here and there, and the ensuing costs in storage space. This may be unavoidable, but it seems almost certain that specialized accessing algorithms can allow these copies to

be virtual, that is, temporarily constructed, interrogated, and discarded only when necessary, so that the long-term storage requirements do not exceed that used for the basic information being represented. Fahlman [1979] has developed algorithms of this sort, but for a slightly different set of data-structures, and without the use of a RMS. I have tried to avoid making design decisions which would rule out algorithms approximating his, for his suggestion of radically parallel database organizations seems very attractive for the long view of information retrieval.

2. *Reorganize the RMS interface:* RMS was designed as an independent subsystem, and in the absence of more comprehensive techniques of control, was vested with a substantial amount of responsibility for choosing among alternate belief revisions, responsibility it should not bear and that this thesis has tried to relieve. The rather haphazard interface between RMS and the decision-making procedures is one result of this. In addition to those questions about RMS suggested for study in [Doyle 1979], the overall organization of RMS should be rationalized in light of its actual role in the larger reasoning program architecture.

3. *Develop convenient syntaxes:* There should be a better syntax to facilitate the input and output of information. This thesis hides some of the ugliest of the reality of using what exists of the program.

4. *Encode information about the world in the database:* I could not even attempt to present an impressive display of the powers of this approach to reasoning because I lack an encoding of a sizable body of information about some problem domain other than the program itself, which is of considerably simpler structure than the rest of the world. Again, I share this problem with others, although there are currently appearing a number of database of facts (but few procedures) about domains.

5. *Encode plans in the plan library:* Of course, this is a subproblem of the previous problem, as any competent program needs not only the facts but know-how.

6. *Catalogue various deliberation procedures:* In addition to encoding the values and the specialized, problem-specific decision procedures of the domains of action in the program, more study

should be applied to develop abstract deliberation procedures in several levels of generality. Rationalization and completion of the library of second-order and higher-order policies seems a primary topic for inquiry, along with investigation of the form of fully recursive or reflective deliberation procedures.

7. *Build a better vocabulary of processes:* The language of the interpreter includes only a rudimentary vocabulary for describing plans and processes. Extensions of this vocabulary depend in part on building up more descriptions of the external world in the database, and in part on the progress of computer science in developing process description languages, parallel and otherwise.

8. *Build a better vocabulary of deliberation:* As a subproblem of the preceding, the vocabulary of actions of policies should be extended.

9. *Develop a vocabulary of advice types:* One aim of this thesis has been to develop mechanisms useful in building a program which can accept, assimilate, and use facts and hint on how to use them. But I have not explored how these pieces of advice might be best conveyed. An important problem involved in realizing a program of this sort is in discovering a vocabulary of advice for imparting facts, values, and skills. For example, informal hints about how to make some decision include advice like (a) choose any one you like, (b) choose quickly, (c) keep in mind that it is raining, and (d) give my suggestion every conceivable consideration or benefit of a doubt. A formal advice vocabulary ought to include formal analogues of these sorts of hints. The problem of advice is closely tied with the discourse understanding problems mentioned below, for humans frequently give procedural or value information as declarative statements, and rely on the advisee to ask and answer questions like What could they have possibly meant by that? and What problem do they think I am facing that that fact would be relevant to?

10. *Apply self-models in hypothetical reasoning:* Many sorts of reasoning processes require the ability to answer questions of ability and other hypotheticals. Many of these questions can be answered by envisioning or predicting the actions and intentions described by the question. One important topic for investigation is that of using the self-description of the program in hypothetical reasoning. Symbolic

execution of the self-description can be used to see what actions would be taken and what their effects would be in certain circumstances, without actually taking the actions or requiring the realization of the circumstances. Symbolic execution involves setting up a sequence (properly, a directed acyclic graph) of temporal situations linked by actions, and asserting the effects of an action in its final situation whenever the preconditions of the action can be proved in its initial situation.[78] In symbolic execution of the self-model, then, the program would create a new state of mind to represent the hypothetical actions. It would then assert the initial conditions in this frame of action, and begin executing within it. Instead of executing its primitives, it would use the specifications of the primitives to assert their effects. The answer to the hypothetical question is then answered by examining this record of symbolic execution.

Symbolic execution of self-models also is valuable in skill introspection and development. Many of the studied techniques for analyzing Lisp programs into plans depend on symbolic execution of the programs and plans. Similarly, the techniques of maintenance and compilation of programs require symbolic execution not only for introspection, but also for compilation of primitives from plans.

11. *Refine the techniques for plan compilation:* One important application of symbolic execution is in compiling refined plans and primitives from other plans and restricting information. Developing the standard compilation techniques (constant folding, dead code elimination, etc.) in this context is an important requirement for the future success of this sort of program. For example, guidelines need to be developed for (a) when to coerce independent steps of a plan into a sequence, (b) when to reduce deliberation to choices or conditionals, (c) when to transform plan variables to local variables or data-structures, and (d) when to substitute subplans or primitives for tasks in a plan.[79]

12. *Study formal historical interpretation:* Collingwood [1946] suggested that the aim of history is not just to record annals, but to discover psychological explanations of the actions of men. This

[78] Shrobe [1979a] gives detailed examples of this technique. See also [Hewitt and Smith 1975].

[79] Burstall and Darlington [1977] and Clark and Sickel [1977] explore program transformations to aid efficiency, and their techniques might be adapted to the plan-compilation task.

involves not only discovering the facts of a situation, but also the ways the participants viewed the situation and the possible actions available to them. That is, the goal of the historian is to infer the attitudes or mental state of each of the participants in the event. The obvious difficulty in this enterprise is the ambiguity of mental states as determined by the recorded physical actions. Even if we have complete annals of the actions of an event, there might have been many completely different mental states of participants which could explain these actions. Was President Nixon an amoral criminal, was he a patriot desperately defending the security of his country, or was he neither of these? To answer questions like this, we must examine all of his actions to see if they are consistent with one interpretation but not another (moderated by an assumption of his rationality). But it may happen that all of our information about his actions is consistent with several interpretations, so that we cannot answer the question.

The program must also make historical analyses of events, for example, to determine just what error was made in some past decision or construction of a procedure when that decision or procedure later leads to an error which must be corrected and avoided in the future. But in this the program also faces ambiguity in reconstructing its past mental states, despite its wealth of records about actions, inferences, and decisions. There are two major sources of this ambiguity. The first is that justification are atemporal records of inferences, so it is difficult to tell just what the set of justifications was at some past time. But even if this problem was overcome, a second source of ambiguity is that a given set of non-monotonic justifications typically admits several interpretations as distinct sets of attitudes. Of course it might be possible to determine which set of attitudes existed from the following actions and inferences, but techniques for making these judgements are completely unexplored. For example, one might think that this problem might be solved by keeping some sort of history list of all inferences and actions. But this cannot work, because these records will be subject to the same insecurity that afflicts other beliefs about the past.[80]

13. *Apply self-models in discourse and multi-agent planning:* One attractive application of hypothetical reasoning by symbolic execution of program models is in using several such models to describe the reasoning faculties and attitudes of other agents for use in cooperative activities like conversations. The proposal here is to employ not just the theory ME, the program's theory of itself, but several copies of ME, one to represent each other person being considered, each copy modified to reflect the differences of that person from the program in its beliefs, desires, values, and skills. Of course the most perspicuous organization of these multiple person models is to have a theory of the prototypical person, describing the common knowledge and skills of people, and to have all other theories be modified copies of this prototype. Each of the particular person models would be used for different people, and further copies of them would be used to represent different people at different times, or in hypothetical situations as mentioned above for ME. Anonymous copies of the prototypical person theory would be used to answer hypothetical questions about the behavior of typical people. Finally, the program might maintain particular person descriptions as its consciences or ideal self-models, so that during deliberation it can query these descriptions to see what is the "right" thing to do (i.e. what would I do if I were perfect?).

How the program might develop such models of its acquaintances from a general person model, or alternatively, develop its general person model from its models of itself and others, are interesting unexplored topics.

14. *Separate the logics of belief, desire, and intention:* In the use of RMS I suggested viewing intentions and other program structures not as embodiments or representations of intentions, but as beliefs of the program about its intentions. This suggestion was motivated by the desire to subsume all

[80] It might seem that this cannot work because the recording of these actions must involve further actions which cannot themselves be recorded on pain of an infinite regress. This may be avoided by having the actions described by the records include the recording substeps as well.

logics of reasoning into the single logic employed by RMS. While this view may be temporarily useful, it may be ultimately misguided. Different attitudes have different logics, and more argument than was presented seems to be necessary for their unification.

A related drawback of this approach of viewing program attitudes as beliefs about attitudes is that it offers a confusion about the "levels" of the program's beliefs. For example, humans sometimes infer that they possess certain attitudes from observations of their actions, as in "I didn't think I wanted to eat, but looking at the amount I put away, I must have been really hungry." But the belief that I desire food in such a case must be different from the desire for several reasons. First, I might be wrong in the inference, in which case my inference would hardly constitute a desire. Second, the reasons for the belief are purely in terms of other beliefs about my actions. But the reasons for a desire will, if the desire is not basic, in general involve both beliefs and desires.

It may be that the particular approach taken in the thesis overcomes these problems, but that is a topic for further investigation. My guess is that the primary error is simply my interpretation of these structures as beliefs about attitudes, rather than the more natural interpretation as the realization of the attitudes themselves. My interpretation stems from a view of RMS as working only with a logic of belief, rather than with several logics for different attitudes. Perhaps the only change necessary is to change the operation of RMS so that it respects these different logics for different attitudes.

Part of this possible confusion between the attitudes seems to stem from an asymmetry between the types of attitudes. Namely, desires and intentions are each represented as statements of the form Desire(content) or Intention(content), where the contents are further concepts. Beliefs, however, are represented as RMS-node(content), where a RMS-node is not a predicate symbol, and content is not a concept. Finding some reorganization or interpretation of these attitudes would go a long way towards cleaning up this problem.

15. *Explore the relations of desires to intentions:* My treatment of desires and intentions and their relations has been most cavalier. The problem of how and when intentions are formed from desires

seems to have received scant study, at least in the parts of the literature on practical reasoning that I have examined.

16. *Investigate multiple loci of consciousness:* We described the consciousness of the program as the perceptions of the interpreter, and unconscious primitives as programs for any other interpreter. But there may be many loci of unconscious action, as the various primitive program interpreters may be distinct machines, as is common practice in the CPU-peripheral organizations of modern computers. We have no similar suggestions about how consciousness might be broken into several loci. We have suggested that there might be several interpreters, each active at different times, and that deliberation procedures can also reflect on the current mental state. Are these properly thought of temporally distinct loci of consciousness? Can consciousness involve simultaneous perceptions of several simultaneously operating subsystems?

### 7.4 Affect, Intellect, and Complex Self-Descriptions

We have presented a model for rational thought which employs only the simplest realizations of a few mental attitudes. While these prove useful for many purposes, the next step is to formalize a wider range of mental attitudes, such as carefulness, confusion, hesitation, and others.[81] Once formalized, these new attitudes may be put to the service of a more powerful reasoner.

Consider the mental attitude *carefulness.* Carefulness has entered experimental AI programs, including this thesis, only in an informal, *ad hoc* way. A program often has two ways of carrying out some

[81] In this section, I have been substantially impressed and motivated by the ideas of Marvin Minsky and Seymour Papert, first in the 1978 draft of their book on the Society of Mind, and later in Minsky's paper on affective exploitation [Minsky 1980]. In addition to exploring the interaction (and in one sense, unity) of affect and intellect, Minsky tries to invert a common conception of affect as complex and intellect as transparent by suggesting that intellectual mechanisms might be built out of simpler affective mechanisms. My suggestions in this section are to study how affect might be built from intellect. At this stage of investigation, my suggestions should not be taken as opposing Minsky's view. Any connection between the two paths of construction is likely to provide ways of building either sort of mental attitude from the other. Where one starts is a matter of convenience. Since this thesis builds up much of the intellectual mechanisms of reasoning, it is most convenient here first to build affect from intellect, and then to build intellect from affect.

activity. Of these, one procedure acts in an automatic fashion, carrying out its steps without pause. The other procedure separates each step with checks to see if it is safe to proceed, that is, whether certain exceptional conditions have arisen from the execution of the previous step. In such cases, common practice is to call the former procedure the normal one, and the second procedure the "careful" version, or "careful mode."

Of course, in these cases the program does not call one procedure careful and the other heedless, it is the programmer who does so. But if the program could also make these discriminations among procedures, its planning and skill development capabilities would be substantially enhanced. When constructing a plan in hierarchical fashion, if the intended result is to be a careful plan, the program might make judgements about which of the steps of the plan must be realized in a careful fashion, and so influence the design of these steps. The program might also deliberately choose to be careful when it judges that it is acting without as much information as it normally prefers, or when it realizes that its actions are likely to be very important or consequential. Thus it would be valuable to formalize some notion of being careful about something so that the program can make decisions about whether to be careful or not, rather than restricting these decisions to the programmer.

Consider *confusion*. This is a very useful attitude to be able to recognize in oneself, for we all use several plans for getting out of confusions. For example, when attempting a difficult project, such as implementing a large program for one's thesis, it is common to try making decision A, postponing it when one gets stuck, working on decision B, postponing B because it seems to depend on first deciding A, working on C until seeing that it depends on the outcome of B, working again on A only to find that it depends on C. From personal experience, I can aver that at this point I realize I'm confused about what to do. What do I do? I apply my realization to think of ways out of my confusion, such as making a graph of the dependencies I perceive among the decisions, and then trying to see if I can make one of the decisions arbitrarily so that I can proceed, and fix it later if it doesn't work out. Of course I try to pick the choice so that fixing it will not be hard, and so that I will make some progress on the other decisions even if the first is wrong, but the main plan is just to make a choice, knowing that it may not be defensible. If I can be more effective in this way because I can recognize and act on my confusion, a program should be able to enjoy the same facility.

Finally, consider *hesitation*. If one can see that one is hesitating about a decision, then that is a valuable consideration in making related choices. In particular, the related choices should be made so that they depend as little as possible on successfully carrying through the hesitant decision. As in the above confusion example, a deadlock breaking decision might be crucial but hesitant, and so its correctness should not be counted on heavily by dependent decisions.

How might we formalize hesitation? Dennett [1978b] suggests the following possibility. He makes a distinction between belief and opinion, where belief is a graded feeling (possibly described by Bayesian evolution rules) upon which action is really based, while opinion, on the other hand, he takes to be all or none assent to linguistic statements. Hesitation (and self-deception) he explains as cases in which one has developed opinions which do not comfortably match one's beliefs. Thus on the basis of a chain of inferences one might make the rational decision to take some action, but since the beliefs involved are not completely certain one has little confidence in the conclusion of the argument, in spite one's avowal that it is the right thing to do. One is willing to declare one's intention, but when it comes down to actually taking the action, one's action, based on the uncertain beliefs rather than on the opinion, does not carry out the intention. Dennett's suggestion might fit into the presently proposed program by identifying what he calls opinions with what I call beliefs, and what he calls beliefs with something derived from policies.

Just as one recognizes complex intellectual attitudes and employs them in deciding what to do, one also recognizes and similarly employs one's emotions. For example, I see myself getting tired and unwilling to continue writing the next chapter of my thesis. To carry through with my intention to finish the chapter by the evening, I carry out a plan which involves imagining how unpleasant prolonged matriculation would be. The plan is based on the expectation that this thought will prove so horrifying that I will resume writing with renewed vigor and determination.

To illustrate these ideas more concretely, but extremely simplistically, consider policies like the following.[82]

```
If the decision is important, prefer to continue deliberating.

If the reasoner is sick, tired, debilitated, mentally impaired, or
otherwise has reason to suspect its mental faculties, try to delay the decision.

If the decision is hateful or distasteful, try to reject the decision.

If the reasoner is angry, frustrated, or confused, try to delay the
decision and relax and reorganize.

If the reasoner is despairing of being able to decide, choose randomly.
```

In these I just try to indicate, without providing any mechanisms for how one might make these judgements, of how affective or emotional considerations might enter into the decision-making process. If one's mind and body are on the fritz, one shouldn't think unless forced to. If one can't stand making the choice, one frequently finds reasons to impugn the choice, to reject it. If one merely finds the choice annoying or distasteful, one just avoids it until time pressure sets in or until it goes away. If one is confused, angry, or frustrated, one delays and relaxes, and perhaps engages in other plans like making lists of options, reorganizing them, etc.

The main point I'm trying to get across here is that if one develops some way of recognizing or observing emotional states by the program looking at itself, then I have sketched how one might proceed to use these sorts of judgements in rational thought, particularly in the ability of thinking rationally about one's own psychological problems. Machines will probably get depressed too, and we ought to figure out how to help them get themselves out of it.

Many emotions may prove useful to a computer program. However, this idea requires much experimentation and study in programs with vastly different forms of their psychologies. Not only will

[82] See [Carbonell 1979] for another approach towards formalizing and using attributions of emotions and complex mental states.

their specific beliefs and skills differ, but the form of their mental lives will differ. There will the purely intellectual programs (Mr. Spock's Revenge), intellects that can hope and fear as well, and perhaps some programs constructed to share the full range of human emotions.

This sort of study would be an ideal laboratory for studying which parts of man's mental life are truly valuable for some purposes, and which parts, if any, are unnecessary accidents of evolution and physiology. Just as geneticists may someday discover enough to allow man to direct his physiological evolution, experimental alien intelligences may help psychologists discover enough to let him direct his psychological evolution as well.[83] On a smaller scale, these experiments may help man improve his repertoire of informal self-analysis and self-help techniques. Given man's age-old desire to direct his future for his own benefit, I see no reason to fear man's obsolescence in the shadow of superintelligent machines. He is much likelier to obsolesce in the shadow of his children.

## 7.5 The Limits and Accuracy of Self-Knowledge

> I have a left shoulder-blade that is a miracle of loveliness. People come miles to see it. My right elbow has a fascination that few can resist.
> Sir W. S. Gilbert, *The Mikado*

How much can the program know about itself? The mechanisms described in this thesis seem to suggest the following directions for investigation.

The program has a model of, in fact direct access to, its nominal sets of concepts, beliefs, assumptions, reasons, desires, intentions, actions, values, and skills. In spite of this, the program can be mistaken about these attitudes because its skills, in particular those comprising the basic operation of the program, need not be fully understood by it for it to be operable. If the program does not correctly understand the details of its own procedures and how they affect its perception of its attitudes, we might

[83] In fact, Wilson [1978] suggests that these two endeavors are more closely connected that is sometimes thought, that if we wish to guide our physical evolution, we must also consider the effects on our psychological evolution.

expect the program to be just as confused about the corrigibility of introspection as we are, for as far as it can tell, it has incorrigible access to all of its mental states. But if the program realizes that it has incomplete or possibly incorrect understandings of its own procedures, then it can conclude that its need not have incorrigible access. The program's access need not be priviliged, for it may run on a computer which displays its entire mental state in a huge bank of lights, and someone watching these lights with an understanding of the design of the computer would be able to tell at least as much about the program as the program itself.

In fact, the program might have a much easier time at introspection than humans, for humans have not clear access and knowledge of their basic mechanisms. It appears possible to give such access and knowledge to a carefully designed machine. A growing literature on program understanding has been concerned with developing techniques for taking a program and analyzing it into its intentional structure. The program under analysis is first converted into its *surface plan*, which simply indicates the data and control flow connections between the parts of the program. This surface plan is then analyzed further into the *design* of the program. The design consists of the *deep plans* underlying the surface plan together with the teleological justifications of the organization and deployment of these plans. Thus the program understanding task takes a program and attempts to infer the decisions and plans that went into its construction, inverting the design process.

The success of this analysis process depends primarily on (1) having a sufficiently rich library of standard plans, and on (2) the program under analysis having some purpose. For the first requirement, Barstow [1977], Rich [1980], and others have developed catalogues of standard programming plans and techniques, and the completion of this task seems to now depend on energies expended in its pursuit rather than on overcoming unsolved problems. For the second requirement, it appears that most analyses can be made successfully using only the information that the program has a purpose, not what that purpose is. de Kleer [1979a,b] found, in the similar task of analyzing electronic circuits into their underlying designs, that almost all analyses succeeded in finding a unique interpretation of the function of the circuit and its components using only the technique of abandoning any interpretation in which some component's function could not be explained. And in those circuits for which multiple possible functions were determined in spite of this heuristic, the circuit usually can be used to perform any of the several functions, and information about the context of use of the circuit suffices to determine which of these interpretations is correct in that context.

The import of these techniques is that they can be embodied in the program just like any other procedures, and in fact, self-applied so that by itself it can determine the structure of all of the LISP functions making up its procedures. If the language in which these plans are phrased is the same as that used by the plans in the plan library, this means that when the program constructs a new procedure, the records left in its design process constitute the analysis of the new procedure. Therefore, the program need not analyze the new procedure further, since its teleological structure is already known.

At the next level of languages down, the program can apply the same techniques with a different vocabulary of surface syntax and plans to understand the machine code implementing these LISP functions. This process can be continued, to give the program an understanding of how machine instructions are implemented in transistors and resistors, how these are implemented in semi-conductors and conductors, and the atomic, even nuclear and subnuclear structure of these, just as humans seek to understand their construction in anatomical, biological, chemical, and physical terms.

In fact, just as the program can modify its own procedures at the highest levels of this chain of implementations, there is no intrinsic barrier (given enough information and suitable sensors and effectors) to the program changing its own construction at these lower levels, for example, by repairing its circuitry, or even transmitting itself to a new computer host (with or without "terminating" its previous "self").[84]

[84] Sandewall [1979] discusses such self-reproduction as a means towards periodically salvaging the useful attitudes and skills of the program. His idea is that the program replaces itself with a "child" made only from all the useful stuff, leaving all the deadwood behind.

What does all this say about the privacy, directness, and corrigibility of the program's self-knowledge? That apparently its mental states need not be any more private than the contents of a normal computer; that it can have direct access to its internal processes to the machine level and not beyond; and that it need not always be correct in its self-understanding. One might build the program in a computer that displays its complete internal state, but neither this nor taps on these lights will permit "mind-reading" without knowledge of the design of the computer and hence the meanings of those states. The program might interpret its ability to change aspects of itself purely by thinking down to the machine language level but not beyond as a difference between mental and physical. And without the procedures to analyze its own procedures or information about the reliability of its hardware, the program can be mistaken about the behavioral import of its consciously visible attitudes. But with such self-analysis procedures and its recorded reasons for attitudes, the program will be able to say, perhaps with much better justification than humans, things like "part of me wants to do this, and part of me says not" by tracing its attitudes through its reasons to its procedures and other attitudes.[85]

[85] Would the construction of artificial intelligence ever occur as a problem to such a program? Perhaps its notion of artificial intelligence would be organically and genetically developed intelligence. Perhaps AI would really be "alternative intelligence."

### 7.6 The Limits of Reason and the Absurd

If when my wife is sleeping
and the baby and Kathleen
are sleeping
and the sun is a flame-white disc
in silken mists
above shining trees, --
if I in my north room
dance naked, grotesquely
before my mirror
waving my shirt round my head
and singing softly to myself:
"I am lonely, lonely.
I was born to be lonely,
I am best so!"
If I admire my arms, my face,
my shoulders, flanks, buttocks
against the yellow drawn shades, --

Who shall say I am not
the happy genius of my household?

William Carlos Williams, *Dance Russe*

This final section discusses some of the most fundamental problems raised by the question "What should I do?" in light of the architecture for a reasoner developed in this thesis. The conclusion is simply a heightening of the paradox of human absurdity to the paradox that absurdity is a consequence of being best at catering to self-significance.

For the program as described to survive, it must matter to itself, it must be self-significant. The actions of the program all involve changing itself. At each step the program packages up its current mental state as a new object, thus entering a new state containing the reified previous state. The program then makes further changes in its sate on the basis of reflecting, on examination of the reified previous state. Its continual question of what to do is always that of how to change its state. (Any effects in the physical world of these mental changes result from the realization of the machine as a physical device with causal connections to the rest of the physical world.) Some changes the program can make in itself can destroy it. For example, it can abandon all its procedures without replacing them by new ones, so that it has no means by which to act in the future. For the program to have some way of preferring other, more sane changes to this one, it must value its own survival. It must be self-significant.

Since the program can self-consciously discuss itself, its survival values can be justified in terms of predicted non-survival. All of the program's attitudes will either appear to it to be premises (depending on no other attitudes), mutually supporting attitudes, or attitudes depending on attitudes of the first two sorts. Indeed, all attitudes may be mutually supporting to some degree if hypotheses can always be inferred from sufficiently many of their consequences. For either premises or mutually supporting attitudes, the program might attempt to find further justify justifications. Such justifications cannot be in terms of other attitudes, or the point of the effort has been missed. The justification also cannot be in terms of the programmer or other external agents, lest the question be begged by merely rephrasing it as a similar question of justification for the external agents. The only sort of answer that seems to be left is a pragmatic one: that doing things one way works (leads to continued survival), and that doing things differently is less certain of working.

For example, Quine invents the metaphor of the web for our systems of beliefs.[86] Our sensory impressions, hypotheses, theories, laws of nature, and laws of reason all populate a great web of belief, beliefs interconnected so that changes in one lead to changes in others, so that any belief may be changed through changes in sufficiently many other beliefs. When confronted with new information, new entries, and changes in the web, we make further changes, either to accommodate or to reject the new entries. Changes most frequently occur in the "sensory" beliefs at the web's periphery, and rarely have repercussions in the web's interior, at the center of which reside the laws of reason and unshakable faiths. In the web metaphor, the only difference between beliefs relevant to their change is the tenacity to which we cling to them, and the tenacity increases as we proceed from the web's periphery to its center. But what is this tenacity of grip on beliefs? Quine suggest that the reasons we hold the beliefs we do are

[86] In section 6 of [Quine 1953].

purely pragmatic, that we change our beliefs so that they lead to successful survival. There is nothing wrong with other changes, it is just that we die if we make them, and along with us ends our web of belief.

But note the form of pragmatic justifications of attitudes: *because holding otherwise leads to non-survival.* To formulate such justifications, the program must be able to realize the possibility of its own non-survival as opposed to it own survival, and hence the possibility of its non-existence as opposed to its existence. This means that the program must be able to view itself as an entity of (possibly) limited temporal duration, and its own life-span as a segment of eternity. It must be able to think of itself as a finite object existing in an infinite eternity, or in traditional terms, *sub specie aeternitatis.*

Here enters the paradox of human absurdity. As an adaptive agent, the program must be self-significant. But as a self-conscious agent, the program can realize its eternal insignificance, and hence a sense of self-insignificance. It sees that while its values make it matter to itself, outside the span of its existence its values have no meaning. Hence the program can see that the way things are does not matter to it if it is not. Further its being not does not matter to eternity, since there are no standards for things mattering to eternity.[87]

Nagel [1979a] phrases this paradox as the result of dragooning transcendent consciousness into the service of mundane existence. Adaptiveness alone may suffice for survival, as is shown by the lower animals and plants. But animals and plants are not absurd, because they are not both self-conscious and adaptive. Only agents both adaptive and self-conscious are absurd, that is, permit the possibility of encountering this paradox of simultaneous self-significance and self-insignificance.

But as this thesis has argued earlier, self-consciousness is necessary for maximal effectiveness in adaptation. Only by self-consciously reflecting on our past and potential actions can we avoid as many pitfalls as possible. Thus absurdity is no accident. The program must adapt to survive, must be self-conscious to be superior at adapting, and hence must be absurd. In the terms of the theory of

[87] Wheeler [1977] and others have suggested that eternity may never exist except when it is possible that some observer, or self-significant agent might exist as part of it.

evolution, the fittest are the absurd.

> And nevertheless I am weary, and I know that there can be no rest for me in the heart of this great city which thinks so much, which has taught me to think, and which forever urges me to think more. And how avoid being excited among all these books which incessantly tempt my curiosity without ever satisfying it? At one moment it is a date I have to look for; at another it is the name of a place I have to make sure of, or some quaint term of which it is important to determine the exact meaning. Words? -- why, yes! words. As a philologist, I am their sovereign; they are my subjects, and, like a good king, I devote my whole life to them. But will I not be able to abdicate some day? I have an idea that there is somewhere or other, quite far from here, a certain little cottage where I could enjoy the quiet I so much need, while awaiting that day in which a greater quiet -- that which can never be broken -- shall come to wrap me all about. I dream of a bench before the threshold and of fields spreading away out of sight. But I must have a fresh smiling young face beside me, to reflect and concentrate all that freshness of nature. I could then imagine myself a grandfather, and all the long void of my life would be filled....
>
> Anatole France, *The Crime of Sylvestre Bonnard*

# REFERENCES

In the following, *IJCAI* refers to one of the *International Joint Conferences on Artificial Intelligence*, held in odd-numbered years.

Allison, G. T., 1971. *Essence of Decision: Explaining the Cuban Missile Crisis*, Boston: Little, Brown.

Anderson, R. M. Jr., 1975. Paradoxes of cosmological self-reference, *Induction, Probability, and Confirmation* (G. Maxwell and R. M. Anderson Jr., eds.), Minneapolis: University of Minnesota Press, 530-540.

Anscombe, G. E. M., 1957. *Intention*, London: Basil Blackwell.

Aristotle, 1962. *Nichomachian Ethics* (M. Ostwald, tr.), Indianapolis: Bobbs-Merrill.

Arrow, K. J., 1967. Values and collective decision-making, *Philosophy, Politics, and Society III* (P. Laslett and W. G. Runciman, eds.), London: Basil Blackwell.

Asimov, I., 1950. *I, Robot*, New York: Gnome Press.

Asimov, I., 1964. *The Rest of the Robots*, New York: Doubleday.

Aune, B., 1977. *Reason and Action*, Dordrecht: D. Reidel.

Austin, J. L., 1962. *How To Do Things With Words*, Cambridge: Harvard University Press.

Backus, J., 1973. Programming language semantics and closed applicative languages, *Proc. Symp. on Principles of Programming Languages*, 71-86.

Barnard, C. I., 1938. *The Functions of the Executive*, Cambridge: Harvard University press.

Barstow, D. R., 1977. Automatic construction of algorithms and data structures using a knowledge base of programming rules, Stanford AI Laboratory, Memo AIM-308.

Barth, J., 1960. *The Sot-Weed Factor*, New York: Doubleday.

Bell, C. G., and Newell, A., 1971. *Computer Structures: Readings and Examples*, New York: McGraw-Hill.

Bell, D., 1976. *The Cultural Contradictions of Capitalism*, New York: Basic Books.

Belnap, N. D., 1976. How a computer should think, *Contemporary Aspects of Philosophy* (G. Ryle, ed.), Stocksfield: Oriel Press, 30-56.

Bennett, J., 1964. *Rationality*, London: Routledge and Kegan Paul.

Bobrow, D. G., and Winograd, T., 1977. An overview of KRL, a knowledge representation language, *Cognitive Science 1*, 3-46.

Boden, M. A., 1977. *Artificial Intelligence and Natural Man*, New York: Basic Books.

Boolos, G., 1979. *The Unprovability of Consistency: An essay in modal logic*, Cambridge: Cambridge University Press.

Borning, A. H., 1979. Thinglab: a constraint-oriented simulation laboratory, Ph.D. thesis, Stanford University, Department of Computer Science.

Brachman, R. J., 1978. A structural paradigm for representing knowledge, Bolt, Beranek, and Newman, Report 3605.

Brachman, R. J., and Smith, B. C., 1980. Special issue on knowledge representation, *ACM Sigart Newsletter 70.*

Brand, M., ed. 1970. *The Nature of Human Action*, Glenview: Scott, Foresman.

Braybrooke, D., and Lindblom, C. E., 1963. *A Strategy of Decision: Policy Evaluation as a Social Process*, New York: Free Press.

Brown, A. L., 1976. Qualitative knowledge, causal reasoning, and the localization of failures, MIT AI Laboratory, TR-362.

Brown, F., 1977. The theory of meaning, University of Edinburgh, Department of Artificial Intelligence Research Report 35.

Brown, F., 1979. A theorem prover for meta theory, *Proc. Fourth Workshop on Automated Deduction*, 155-160.

Burstall, R. M., and Darlington, J., 1977. A transformation system for developing recursive programs, *J. ACM* 24, 44-67.

Camus, A., 1955. *The Myth of Sisyphus and other essays*, New York: Random House.

Carbonell, J. G., 1979. Computer models of human personality traits, Carnegie-Mellon University, Computer Science Department, CS-79-154.

Carnegie, D., 1936. *How to Win Friends and Influence People*, New York: Simon and Schuster.

Carnegie, D., 1944. *How to Stop Worrying and Start Living*, New York: Simon and Schuster.

Cartwright, R., and McCarthy, J., 1979. Recursive programs as functions in a first-order theory, Stanford CSD Report 79-717.

Castaneda, H.-N., 1975. *Thinking and Doing*, Dordrecht: D. Reidel.

Cattell, R. G. G., 1978. Formalization and automatic derivation of code generators, Carnegie-Mellon University, Computer Science Department.

Chandler, A. D. Jr., 1962. *Strategy and Structure: Chapters in the History of the Industrial Enterprise*, Cambridge: MIT Press.

Charniak, E., Riesbeck, C., and McDermott, D., 1979. *Artificial Intelligence Programming*, Baltimore: L. E. Erlbaum.

Chisholm, R., 1978. Practical reason and the logic of requirement, *Practical Reasoning* (J. Raz, ed.), Oxford: Oxford University Press, 118-127.

Church, A., 1941. The calculi of lambda-conversion, *Annals of Mathematics Studies 6*, Princeton.

Clark, K., and Sickel, S., 1977. Predicate logic: a calculus for deriving programs, *IJCAI-77*, 410-411.

Cohen, P. R., 1978. On knowing what to say: planning speech acts, Department of Computer Science, University of Toronto, TR-118.

Collingwood, R. G., 1946. *The Idea of History*, Oxford: Oxford University Press.

Collins, A., 1978. Fragments of a theory of human plausible reasoning, *Proc. Second Conf. Theoretical Issues in Natural Language Processing*, 194-201.

Dacey, R., 1978. A theory of conclusions, *Philosophy of Science 45*, 563-574.

Davis, L. H., 1979. *Theory of Action*, Englewood Cliffs: Prentice-Hall.

Davis, R., 1976. Applications of meta level knowledge to the construction, maintainance and use of large knowledge bases, Stanford AI Laboratory, Memo AIM-283.

Davis, R., 1980. Meta-rules: reasoning about control, MIT AI Laboratory, Memo 576.

Dennett, D. C., 1969. *Content and Consciousness*, London: Routledge and Kegan Paul.

Dennett, D. C., 1978a. Current issues in the philosophy of mind, *Amer. Phil. Quarterly 15*, 249-261.

Dennett, D. C., 1978b. How to change your mind, *Brainstorms*, Montgomery, Vermont: Bradford Books, 300-309.

Dennett, D. C., 1978c. *Brainstorms*, Montgomery, Vermont: Bradford Books.

de Kleer, J., 1979a. Causal and teleological reasoning in circuit recognition, MIT AI Laboratory, TR-529.

de Kleer, J., 1979b. The origin and resolution of ambiguities in causal arguments, *IJCAI-79*, 197-203.

de Kleer, J., Doyle, J., Steele, G. L. Jr., and Sussman, G. J., 1977. Explicit control of reasoning, *Proc. ACM Symp. on Artificial Intelligence and Programming Languages*, Rochester, New York, also MIT AI Laboratory, Memo 427.

de Kleer, J., and Harris, G., 1979. Truth maintenance systems in problem solving, draft, Xerox PARC.

Doyle, J., 1976. The use of dependency relationships in the control of reasoning, MIT AI Laboratory, Working Paper 133.

Doyle, J., 1977. Hierarchy in knowledge representations, MIT AI Laboratory, Working Paper 159.

Doyle, J., 1979. A truth maintenance system, *Artificial Intelligence 12*, 231-272.

Doyle, J., and London, P., 1980. A selected descriptor-indexed bibliography to the literature on belief revision, *SIGART Newsletter 71*.

Dray, W. H., 1964. *Philosophy of History*, Englewood Cliffs: Prentice-Hall.

Dreyfus, H. L., 1979. *What Computers Can't Do*, revised ed., New York: Harper and Row.

Drucker, P. F., 1946. *Concept of the Corporation*, New York: John Day.

Drucker, P. F., 1974. *Management: Tasks, Responsibilities, Practices*, New York: Harper and Row.

Duda, R. O., Hart, P. E., and Nilsson, N. J., 1976. Subjective bayesian methods for rule-based inference systems, *Proc. National Computer Conference, AFIPS Conference Proceedings Vol. 45*, 1075-1082.

Dummett, M. A. E., 1973. The justification of deduction, *Proc. British Academy*, Vol. LIX.

Edgley, R., 1969. *Reason in Theory and Practice*, London: Hutchinson.

Ellis, A. and Harper, R. A., 1961. *A Guide to Rational Living*, Englewood Cliffs: Prentice-Hall.

Enters, H., 1924. An meine Geschwister in Deutschland, *Die Kleine, muehselige Welt des jungen Hermann Enters* (K. Eckert, ed.), Wuppertal: Born-Verlag 1970.

Ernst, G. W., and Newell, A., 1969. *GPS: A Case Study in Generality and Problem Solving*, New York: Academic Press.

Fahlman, S. E., 1974. A planning system for robot construction tasks, *Artificial Intelligence 5*, 1-49.

Fahlman, S. E., 1979. *NETL: A System for Representing and Using Real World Knowledge*, Cambridge: MIT Press.

Feferman, S., 1960. Arithmetization of metamathematics in a general setting, *Fund. Math. LXIX*, 53.

Fikes, R. E., 1972. Monitored execution of robot plans produced by STRIPS, *IFIP 1971*, Amsterdam: North-Holland, 189-194.

Fikes, R. E., 1975. Deductive retrieval mechanisms for state description models, *IJCAI-75*, 99-106.

Fikes, R. E., Hart, P. E., and Nilsson, N. J., 1972. Learning and executing generalized robot plans, *Artificial Intelligence 3*, 251-288.

Fikes, R. E., and Nilsson, N. J., 1971. STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence 2*, 189-208.

Fodor, J. A., 1968. *Psychological Explanation*, New York: Random House.

Fodor, J. A., 1975. *The Language of Thought*, New York: Crowell.

Fodor, J. A., 1978. Methodological solipsism as a research strategy in psychology, MIT Department of Psychology, draft.

Fox, M. S., 1978. Knowledge structuring: an overview, *Proc. Second Conf. Canadian Society for Computational Studies of Intelligence*, 146-155.

Fox, M. S., 1979. Organization structuring: designing large complex software, Carnegie-Mellon University, Computer Science Department, CS-79-155.

Freud, S., 1937. *The Interpretation of Dreams* (A. A. Brill, tr.), New York: Macmillan.

Gaines, B. R., 1976. Foundations of fuzzy reasoning, *International Journal of Man-Machine Studies 8*, 623-668.

Gardiner, P., 1974. *The Philosophy of History*, London: Oxford University Press.

Gauthier, D. P., 1963. *Practical Reasoning*, London: Oxford University Press.

Giles, R., 1976. A logic for subjective belief, *Foundations of Probability Theory, Statistical Inference, and Statistical Theories of Science*, Vol. 1, (W. L. Harper and C. A. Hooker, eds.), Dordrecht: Reidel, 41-72.

Glover, J., 1976. *The Philosophy of Mind*, Oxford: Oxford University Press.

Godel, K., 1931. On formally undecidable propositions of *Principia Mathematica* and related systems I, *From Frege to Godel: A Source Book in Mathematical Logic, 1879-1931* ( J. van Heijenoort, ed.), Cambridge: Harvard University Press, 1967, 596-616.

Goldman, A. I., 1970. *A Theory of Human Action*, Princeton: Princeton University Press.

Goldstein, I. P., 1975. Summary of MYCROFT: a system for understanding simple picture programs, *Artificial Intelligence 6*, 249-288.

Good, I. J., 1952. Rational decisions, *Journal of the Royal Statistical Society B 14*, 107-114.

Goodman, N., 1973. The problem of counterfactual conditionals, *Fact, Fiction, and Forecast*, third edition, New York: Bobbs-Merrill, 3-27.

Gordon, M., Milner, R., Morris, L., Newey, M., and Wadsworth, C., 1978. A metalanguage for interactive proof in LCF, *Proc. Fifth Symposium on Principles of Programming Languages*, 199-130.

Green, C., 1969. Theorem-proving by resolution as a basis for question-answering systems, *Machine Intelligence 4* (B. Meltzer and D. Michie, eds.), New York: American Elsevier, 183-205.

Grice, H. P., 1969. Utterer's meaning and intentions, *Philosophical Review*, 147-177.

Grosz, B. J., 1979. Utterance and objective: issues in natural language processing, *IJCAI-79*, 1067-1076.

Gustafson, D. F., ed., 1964. Essays in Philsophical Psychology, New York: Anchor.

Haack, S., 1978. *Philosophy of Logics*, Cambridge: Cambridge University Press.

Hare, R. M., 1952. *The Language of Morals*, Oxford: Oxford University Press.

Hare, R. M., 1963. *Freedom and Reason*, Oxford: Oxford University Press.

Harel, D., 1979. *First Order Dynamic Logic*, Berlin: Springer-Verlag.

Harman, G., 1973. *Thought*, Princeton: Princeton University Press.

Harman, G., 1976. Practical reasoning, *Review of Metaphysics XXIX*, 431-463.

Harman, G., 1977. *The Nature of Morality*, New York: Oxford University Press.

Harper, W. L., 1976. Rational belief change, Popper functions, and counterfactuals, *Foundations of Probability Theory, Statistical Inference, and Statistical Theories of Science*, Vol. 1, (W. L. Harper and C. A. Hooker, eds.), Dordrecht: Reidel, 73-115.

Harrison, A., 1978. *Making and Thinking: A Study of Intelligent Activities*, Indianapolis: Hackett.

Hayes, P. J., 1970. Robotologic, *Machine Intelligence 5* (B. Meltzer and D. Michie, eds.), New York: American Elsevier, 533-554.

Hayes, P. J., 1971. A logic of actions, *Machine Intelligence 6* (B. Meltzer and D. Michie, eds.), New York: American Elsevier, 495-520.

Hayes, P. J., 1973a. The frame problem and related problems in artificial intelligence, *Artificial and Human Thinking* (A. Elithorn and D. Jones, eds.), San Francisco: Josey-Bass.

Hayes, P. J., 1973b. Computation and deduction, *Proc. MFCS Symposium*, Czech. Acad. of Sciences, 105-117.

Hayes, P. J., 1974. Some problems and non-problems in representation theory, *Proc. Conf. Artificial Intelligence and Simulation of Behavior*, 63-79.

Hayes, P. J., 1977a. In defence of logic, *IJCAI-77*, 559-565.

Hayes, P. J., 1977b. The logic of frames, Department of Computer Science, University of Essex.

Hayes, Ph. J., 1977. On semantic nets, frames, and associations, *IJCAI-75*, 99-107.

Hayes-Roth, F., and Lesser, V. R., 1977. Focus of attention in the Hearsay-II speech understanding system, Computer Science Department, Carnegie-Mellon University.

van Heijenoort, J., ed., 1967. *From Frege to Godel: A Source Book in Mathematical Logic, 1879-1931*, Cambridge: Harvard University Press.

Heinlein, R. A., 1966. *The Moon is a Harsh Mistress*, New York: G. P. Putnam and Sons.

Hendrix, G. G., 1975. Expanding the utility of semantic networks through partitioning, *IJCAI-75*, 115-121.

Hewitt, C. E., 1972. Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot, MIT AI Laboratory, TR-258.

Hewitt, C. E., 1977. Viewing control structures as patterns of passing messages, *Artificial Intelligence 8*, 323-364.

Hewitt, C. E., and Smith, B., 1975. Towards a programming apprentice, *IEEE Transactions on Software Engineering SE-1*, 26-45.

Heyting, A., 1956. *Intuitionism: An Introduction*, Amsterdam: North-Holland.

Hilbert, D., 1925. On the infinite, *From Frege to Godel: A Source Book in Mathematical Logic, 1879-1931* ( J. van Heijenoort, ed.), Cambridge: Harvard University Press, 1967, 367-392.

Hilpinen, R., (ed.) 1971. *Deontic Logic: Introductory and Systematic Readings*, Dordrecht: Reidel.

Hintikka, J., 1962. *Knowledge and Belief*, Ithica: Cornell University Press.

Hofstader, D. R., 1979. *Godel, Escher, Bach: An Eternal Golden Braid*, New York: Basic Books.

Hook, S., ed., 1963. *Philosophy and History*, NY: New York University Press.

James, W., 1971. The will to believe, *Reason and Responsibility* (Feinberg, ed.), Encino: Dickenson, 83-90.

Johnson, S. M., 1977. *First Person Singular: Living the Good Life Alone*, New York: Lippincott.

Kenny, A. J. P., 1978. Practical reasoning and rational appetite, *Practical Reasoning* (J. Raz, ed.), Oxford: Oxford University Press, 63-80.

Kierkegaard, S., 1944. *The Concept of Dread* (W. Lowrie, ed.), Princeton: Princeton University Press.

Kleene, S. C., 1950. *An Introduction to Metamathematics*, Princeton: Van Nostrand.

Kornfeld, W. A., 1979. ETHER - a parallel problem solving system, *IJCAI-79*, 490-492.

Kowalski, R., 1974. Logic for problem solving, University of Edinburgh, Department of Artificial Intelligence, DCL memo 75.

Kramosil, I., 1975. A note on deduction rules with negative premises, *IJCAI-75*, 53-56.

Kreisel, G., 1968. A survey of proof theory, *J. Symbolic Logic 33*, 321-388.

Kreisel, G., 1971. A survey of proof theory II, *Proc. Second Scandinavian Logic Symposium* (J. E. Fenstad, ed.), Amsterdam: North-Holland, 109-170.

Kreisel, G., 1977. On the kind of data needed for a theory of proofs, *Logic Colloquium 76* (R. Gandy and M. Hyland, eds.), Amsterdam: North-Holland, 111-128.

Kripke, S. A., 1975. "Outline of a Theory of Truth," *Journal of Philosophy*, 72, 690-716.

Lakatos, I., 1976. *Proofs and Refutations: the logic of mathematical discovery* (J. Worrall and E. Zahar, eds.), Cambridge: Cambridge University Press.

Latombe, J.-C., 1976. Artificial intelligence in computer-aided design: the "TROPIC" system, SRI, Technical Note 125.

Latombe, J.-C., ed., 1978. *Artificial Intelligence and Pattern Recognition in Computer-Aided Design*, Amsterdam: North-Holland.

Latombe, J.-C., 1979. Failure processing in a system for designing complex assemblies, *IJCAI-79*, 508-515.

Lehrer, K., and Paxson, T. Jr., 1969. Knowledge: undefeated justified true belief, *Journal of Philosophy LXVI*, 225-237.

Lehrer, K., 1974. *Knowledge*, Oxford: Oxford University Press.

Lenat, D. B., 1977. The ubiquity of discovery, *Artificial Intelligence 9*, 257-285.

Lewis, D., 1973. *Counterfactuals*, Cambridge: Harvard University Press.

Linsky, L., 1971. *Reference and Modality*, Oxford: Oxford University Press.

Linsky, L., 1977. *Names and Descriptions*, Chicago: University of Chicago Press.

London, P. E., 1978. Dependency networks as a representation for modelling in general problem solvers, Department of Computer Science, University of Maryland, TR-698.

March, J. G., and Simon, H. A., 1958. *Organizations*, New York: Wiley.

Martin, W. A., 1979. Philosophical foundations for a linguistically oriented semantic network, MIT Laboratory for Computer Science, draft.

McAllester, D. A., 1978. A three-valued truth maintenance system, MIT AI Laboratory, Memo 473.

McCarthy, J., 1958. Programs with common sense, reprinted in *Semantic Information Processing* (M. Minsky, ed.), Cambridge: MIT Press (1968), 403-410.

McCarthy, J., et al, 1965. *LISP 1.5 Programmer's Manual*, Cambridge: MIT Press.

McCarthy, J., and Hayes, P. J., 1969. Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence 4* (B. Meltzer and D. Michie, eds.), New York: American Elsevier, 463-502.

McDermott, D., 1978. Planning and acting, *Cognitive Science 2*, 71-109.

McDermott, D., 1980. Non-monotonic logic II: non-monotonic modal theories, Yale University, Department of Computer Science, Report 174.

McDermott, D., and Doyle, J., 1978. Non-monotonic logic I, MIT AI Laboratory, Memo 486.

McDermott, J., and Forgy, C., 1976. Production system conflict resolution strategies, Computer Science Department, Carnegie-Mellon University.

McKeeman, W. M., Horning, J. J., and Wortman, D. B., 1970. *A Compiler Generator*, Englewood Cliffs: Prentice-Hall.

Miller, G. A., Galanter, E., and Pribram, K., 1960. *Plans and the Structure of Behavior*, New York: Holt.

Miller, M. L., 1979. Planning and debugging in elementary programming, Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science.

Minsky, M., 1965. Matter, mind, and models, *Proc. of the IFIP Congress*, 45-49.

Minsky, M., 1974. A framework for representing knowledge, MIT AI Laboratory, Memo 306, and (without appendix) *The Psychology of Computer Vision* (P. Winston, ed.), New York: McGraw-Hill, 1975.

Minsky, M., 1977. Plain talk about neurodevelopmental epistemology, *IJCAI-77*, 1083-1092.

Minsky, M., 1979. K-lines: a theory of memory, MIT AI Laboratory, Memo 516.

Minsky, M., 1980. Affective exploitation: a view of emotion and intellect, MIT AI Laboratory, draft.

Minsky, M., and Papert, S., 1973. *Artificial Intelligence*, Eugene, Oregon: Condon Lecture Publications.

Minsky, M., and Papert, S., 1978. *The Society Theory of Mind*, MIT AI Laboratory, draft.

Montague, R., 1963. Syntactical treatments of modality, with corollaries on reflection principles and finite axiomatizability, *Acta Philosophica Fennica*, 16, 153-167.

Moore, R. C., 1979. Reasoning about knowledge and action, Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science.

Nagel, T., 1970. *The Possibility of Altruism*, Princeton: Princeton University Press.

Nagel, T., 1979a. The absurd, *Mortal Questions*, Cambridge: Cambridge University Press, 11-23.

Nagel, T., 1979b. The fragmentation of value, *Mortal Questions*, Cambridge: Cambridge University Press, 128-141.

Nagel, T., 1979c. Brain bisection and the unity of consciousness, *Mortal Questions*, Cambridge: Cambridge University Press, 147-164.

Nagel, T., 1979d. What is it like to be a bat?, *Mortal Questions*, Cambridge: Cambridge University Press, 165-180.

Newell, A., 1969. Heuristic programming: ill-structured problems, *Progress in Operations Research, Vol. III* (Aronofsky, ed.), 360-414.

Newell, A., and Simon, H. A., 1963. GPS, a program that simulates human thought, *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), New York: McGraw-Hill, 279-293.

Nilsson, N. J., 1980. *Principles of Artificial Intelligence*, Palo Alto: Tioga.

Norman, R., 1971. *Reasons for Actions*, New York: Barnes and Noble.

Nozick, R., 1974. *Anarchy, State and Utopia*, New York: Basic Books.

Pascal, B., 1971. The wager, from *Pensees*, in *Reason and Responsibility* (Feinberg, ed.), Encino: Dickenson, 81-83.

Perrault, C. R., Allen, J. F., and Cohen, P. R., 1978. Speech acts as a basis for understanding dialogue coherence, *Proc. Second Conf. Theoretical Issues in Natural Language Processing*, 125-132.

Post, E. L., 1943. Formal reductions of the general combinatorial decision problem, *Am. J. Math. 65*, 197-268.

Pratt, V. R., 1977. The competence/performance dichotomy in programming, MIT AI Laboratory, Memo 400.

Prawitz, D., 1973. Towards a foundation of general proof theory, *Logic, Methodology, and Philosophy of Science IV* (P. Suppes, L. Henkin, A. Joja, Gr. C. Moisil, eds.), Amsterdam: North-Holland, 225-250.

Putnam, H., 1975. Philosophy and our mental life, *Mind, Language, and Reality*, Cambridge: Cambridge University Press, 291-303.

Putnam, H., 1978. Truth and reason, *Reason and History*, draft.

Quine, W. V., 1953. Two dogmas of empiricism, *From a Logical Point of View*, Cambridge: Harvard University Press.

Quine, W. v., 1966. *The Ways of Paradox and other essays*, Cambridge: Harvard University Press.

Quine, W. V., 1970. *Philosophy of Logic*, Englewood Cliffs: Prentice-Hall.

Quine, W. V., and Ullian, J. S., 1978. *The Web of Belief*, second edition, New York: Random House.

Rabin, M. O., 1974. Theoretical impediments to artificial intelligence, *Information Processing 74*, Amsterdam: North-Holland, 615-619.

Rawls, J., 1971. *A Theory of Justice*, Cambridge: Harvard University Press.

Raz, J., 1978. *Practical Reasoning*, Oxford: Oxford University Press.

Rescher, N., 1964. *Hypothetical Reasoning*, Amsterdam: North Holland.

Rescher, N., 1966. *The Logic of Commands*, London: Routledge and Kegan Paul.

Rescher, N., 1968. *Topics in Philosophical Logic*, Dordrecht: D. Reidel.

Rescher, N., 1976. *Plausible Reasoning*, Amsterdam: Van Gorcum.

Rescher, N., and Urquhart, A., 1971. *Temporal Logic*, New York: Springer-Verlag.

Reiter, R., 1978. On reasoning by default, *Proc. Second Conf. Theoretical Issues in Natural Language Processing*, 210-218.

Reiter, R., 1979. A logic for default reasoning, Department of Computer Science, University of British Columbia, TR-79-8.

Resnik, M. D., 1974. On the philosophical significance of consistency proofs, *J. Phil. Logic 3*, 133-147.

Reynolds, J., 1972. Definitional interpreters for higher order programming languages, *ACM Annual Conference Proceedings.*

Rich, C., 1980. Inspection methods in programming, Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science.

Rich, C., and Shrobe, H. E., 1976. Initial report on a LISP programmer's apprentice, MIT AI Laboratory, TR-354.

Rich, C., Shrobe, H. E., and Waters, R. C., 1979. Computer aided evolutionary design for software engineering, MIT AI Laboratory, Memo 506.

Richards, D. A. J., 1971. *A Theory of Reasons for Action*, London: Oxford University Press.

Rosenberg, J. F., 1978. *The Practice of Philosophy*, Englewood Cliffs: Prentice-Hall.

Rosenberg, R. L., 1980. Incomprehensible computer systems: knowledge without wisdom, MIT Laboratory for Computer Science, TR-227.

Ross, W. D., 1930. *The Right and the Good*, Oxford: Oxford University Press.

Rubin, A. D., 1975. Hypothesis formation and evaluation in medical diagnosis, MIT AI Laboratory, TR-316.

Russell, B., 1908. Mathematical logic as based on the theory of types, *From Frege to Godel: A Source Book in Mathematical Logic, 1879-1931* ( J. van Heijenoort, ed.), Cambridge: Harvard University Press, 1967, 150-182.

Russell, B., 1930. *The Conquest of Happiness*, New York: Liveright.

Rychner, M. D., 1976. Production systems as a programming language for artificial intelligence applications, 3 Volumes, Computer Science Department, Carnegie-Mellon University.

Ryle, G., 1949. *The Concept of Mind*, London: Hutchinson.

Sacerdoti, E. D., 1974. Planning in a hierarchy of abstraction spaces, *Artificial Intelligence 5*, 115-135.

Sacerdoti, E. D., 1977. *A Structure for Plans and Behavior*, New York: American Elsevier.

Sacerdoti, E. D., 1979. Problem solving tactics, *IJCAI-79*, 1077-1085.

Sandewall, E., 1979. Biological software, *IJCAI-79*, 744-747.

Sartre, J.-P., 1956. *Being and Nothingness* (H. Barnes, tr.), New York: Philosophical Library.

Schank, R. C., 1979. Interestingness: controlling inferences, *Artificial Intelligence 12*, 273-297.

Schmidt, C. F., Sridharan, N. S., and Goodson, J. L., 1978. The plan recognition problem: an intersection of psychology and artificial intelligence, *Artificial Intelligence 11*, 45-83.

Schwartz, S. P., ed. 1977. *Naming, Necessity, and Natural Kinds*, Ithica: Cornell University Press.

Scott, D., 1973. Models for various type-free calculi, *Logic, Methodology and Philosophy of Science IV* (P. Suppes, L. Henkin, A. Joja, Gr. C. Moisil, eds.), Amsterdam: North-Holland.

Scriven, M., 1959. Truisms as the grounds for historical explanations, *Theories of History* (P. Gardiner, ed.), New York: Free Press of Glencoe, 443-475.

Scriven, M., 1963. New issues in the logic of explanation, *Philosophy and History* (S. Hook, ed.), New York: New York University Press, 339-361.

Searle, J. R., 1969. *Speech Acts*, Cambridge: Cambridge University Press.

Searle, J. R., 1978. *Prima facie* obligations, *Practical Reasoning* (J. Raz, ed.), Oxford: Oxford University Press, 81-90.

Searle, J. R., 1979. The intentionality of intention and action, *Inquiry 22*, 253-280.

Searle, J. R., 1980. Notes on artificial intelligence, *Behavioral and Brain Sciences*, to appear.

Shaffer, J. A., 1968. *Philosophy of Mind*, Englewood Cliffs: Prentice-Hall.

Shrobe, H. E., 1979a. Dependency directed reasoning for complex program understanding, MIT AI Laboratory, TR-503.

Shrobe, H. E., 1979b. Explicit control of reasoning in the programmer's apprentice, *Proc. Fourth Workshop on Automated Deduction*, 97-102.

Simon, H. A., 1969. *The Sciences of the Artificial*, Cambridge: MIT Press.

Simon, H. A., 1976. *Administrative Behavior*, third ed., New York: Free Press.

Smith, B. C., 1978. Levels, layers, and planes: the framework of a theory of knowledge representation semantics, Masters thesis, MIT Electrical Engineering and Computer Science.

Smith, R. G., and Davis, R., 1978. Distributed problem solving: the contract net approach, *Proc. Second Conf. Canadian Society for Computational Studies of Intelligence*, 278-287.

Smullyan, R. M., 1957. Languages in which self-reference is possible, *J. Symb. Logic*, 22, 55-67.

Smullyan, R. M., 1978. *What is the Name of this Book? The Riddle of Dracula and other Logical Puzzles*, Englewood Cliffs: Prentice-Hall.

Smullyan, R. M., 1980. *This Book Needs No Title*, Englewood Cliffs: Prentice-Hall.

Sosa, E., 1975. *Causation and Counterfactuals*, London: Oxford University Press.

Sridharan, N. S., 1976. The frame and focus problems in AI: discussion in relation to the BELIEVER system, *Proc. Conf. Artificial Intelligence and Simulation of Behavior*, 322-333.

Sridharan, N. S., and Hawrusik, F., 1977. Representation of actions that have side-effects, *IJCAI-77*, 265-266.

Stallman, R. M., and Sussman, G. J., 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence 9*, 135-196.

Steele, G. L. Jr., and Sussman, G. J., 1976. LAMBDA: the ultimate imperative, MIT AI Laboratory, Memo 353.

Steele, G. L. Jr., and Sussman, G. J., 1978a. The revised report on SCHEME, a dialect of LISP, MIT AI Laboratory, Memo 452.

Steele, G. L. Jr., and Sussman. G. J., 1978b. The art of the interpreter, or the modularity complex, MIT AI Laboratory, Memo 453.

Steele, G. L. Jr., and Sussman, G. J., 1978c. Constraints, MIT AI Laboratory, Memo 502.

Stefik, M. J., 1980. Planning with constraints, Stanford University, Computer Science Department, Report STAN-CS-80-784.

Strawson, P. F., 1967. *Philosophical Logic*, Oxford: Oxford University Press.

Suppes, P., 1957. *Introduction to Logic*, New York: Van Nostrand.

Suppes, P., 1967. Decision theory, *The Encyclopedia of Philosophy, Vol. II* (P. Edwards, ed.), New York: Macmillan, 310-314.

Suppes, P., 1977. A survey of contemporary learning theories, *Foundational Problems in the Special Sciences* (R. E. Butts and J. Hintikka, eds.), Dordrecht: Reidel, 217-239.

Sussman, G. J., 1975. *A Computer Model of Skill Acquisition*, New York: American Elsevier.

Sussman, G. J., and McDermott, D., 1972. From PLANNER to CONNIVER - a genetic approach, *Proc. AFIPS FJCC*, 1171-1179.

Tarski, A., 1944. The semantic conception of truth and the foundations of semantics, *Philosophy and Phenomenological Research IV*, 3, 341-375.

Tate, A., 1975. Interacting goals and their use, *IJCAI-75*, 215-218.

Tate, A., 1977. Generating project networks, *IJCAI-77*, 888-893.

Taylor, R., 1966. *Action and Purpose*, Englewood Cliffs: Prentice-Hall.

Taylor, R., 1974. *Metaphysics*, second ed., Englewood Cliffs: Prentice-Hall.

Thompson, A., 1979. Network truth maintenance for deduction and modelling, *IJCAI-79*, 877-879.

Tinbergen, N., 1951. *The Study of Instinct*, Oxford: Clarendon Press.

Tukey, J. W., 1960. Conclusions vs decisions, *Technometrics 2*, 423-433.

Turing, A. M., 1936. On computable numbers with an application to the entscheidungsproblem, *Proc. London Math. Soc. Ser. 2, 42*, 230-265.

Turner, R., 1978. Counterfactuals without possible worlds, Department of Computer Science, University of Essex.

Wason, P. C., and Johnson-Laird, P. N., 1972. *Psychology of Reasoning: Structure and Content*, Cambridge: Harvard University Press.

Weinreb, D., and Moon, D., 1979. Lisp machine manual, MIT AI Laboratory.

Weizenbaum, J., 1976. *Computer Power and Human Reason*, San Francisco: W. H. Freeman.

Weyhrauch, R. W., 1978. Prolegomena to a theory of mechanized formal reasoning, Stanford AI Laboratory, AIM-315.

Wheeler, J. A., 1977. Genesis and observership, *Foundational Problems in the Special Sciences* (R. E. Butts and J. Hintikka, eds.), Dordrecht: D. Reidel, 3-33.

White, A. R., 1968. *The Philosophy of Action*, Oxford: Oxford University Press.

Wiest, J. D., and Levy, F. K., 1977. *A Management Guide to PERT/CPM: with GERT/PDM/DCPM and other Networks*, second edition, Englewood Cliffs: Prentice-Hall.

Wilensky, R., 1978. Understanding goal-based stories, Yale University, Department of Computer Science, Report 140.

Wilks, Y., and Bien, J., 1979. Speech acts and multiple environments, *IJCAI-79*, 968-970.

Wilson, E. O., 1978. *On Human Nature*, Cambridge: Harvard University Press.

Winston, P. H., 1975. Learning structural descriptions from examples, *The Psychology of Computer Vision* (P. H. Winston, ed.), New York: McGraw-Hill, 157-209.

Yessenin-Volpin, A. S., 1970. The ultra-intuitionistic criticism and the antitraditional program for foundations of mathematics, *Intuitionism and Proof Theory* (Proc. Conf. Buffalo, NY, 1968), Amsterdam: North-Holland, 3-45.

Zadeh, L., 1975. Fuzzy logic and approximate reasoning, *Synthese 30*, 407-428.

**CS-TR Scanning Project**
**Document Control Form**

Date : 2 /22 /96

**Report #** AI-TR-581

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
□ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) □ Technical Memo (TM)
□ Other:

## Document Information

Number of pages: 254 (262-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
□ Single-sided or
☒ Double-sided

Intended to be printed as :
□ Single-sided or
□ Double-sided

Print type:
□ Typewriter ☒ Offset Press □ Laser Print
□ InkJet Printer □ Unknown □ Other:

Check each if included with document:

☒ DOD Form (2) ☒ Funding Agent Form ☒ Cover Page
□ Spine □ Printers Notes □ Photo negatives
□ Other:

Page Data:

Blank Pages (by page number):

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description : Page Number:
IMAGE MAP! (1-254 ) 1, UN#'ED BLANK, 2, UN#BLANK,
3-7, UN# BLANK, 8, UN#BLK,
9, UN#BLK, 10-249
(255-262 ) SCANCONTROL, COVER, FUNDING AGENT, DOD (2), TRGT'S (3)

Scanning Agent Signoff:
Date Received: 2/22/96 Date Scanned: 2/23/96 Date Returned: 2/29/96

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DS/LCS Document Control Form cstrform.vsd

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

**REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
TR-581

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (*and Subtitle*)
A Model for Deliberation, Action and Introspection

5. TYPE OF REPORT & PERIOD COVERED
Technical Report

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(*s*)
Jon Doyle

8. CONTRACT OR GRANT NUMBER(*s*)
N00014-75-C-0643
MCS77-04828

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Advanced Research Projects Agency
1400 Wilson Blvd
Arlington, Virginia 22209

12. REPORT DATE
May 1980

13. NUMBER OF PAGES
249

14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)
Office of Naval Research
Information Systems
Arlington, Virginia 22217

15. SECURITY CLASS. (*of this report*)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Distribution of this document is unlimited.

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

18. SUPPLEMENTARY NOTES

None

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

| | | |
|---|---|---|
| Artificial Intelligence | Deliberate Action | Execution |
| Reasoning | Decision-making | Consciousness |
| Problem-solving | Control | Introspection |
| Knowledge Representation | Planning | Belief Revision |

(more)

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

This thesis investigates the problem of controlling or directing the reasoning and actions of a computer program. The basic approach explored is to view reasoning as a species of action, so that a program might apply its reasoning powers to the task of deciding what inferences to make as well as deciding what other actions to take. A design for the architecture of reasoning programs is proposed. This architecture involves self-consciousness, intentional actions, deliberate adaptions, and a form of

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

#19. Learning
Non-monotonic Logic
Defeasible Reasons
Dialectical Argument
Desire
Intention

#20.

decision making based on dialectical argumentation. A program based on this architecture inspects itself, describes aspects of itself to itself, and uses this self-reference and these self-descriptions in making decisions and taking actions. The program's mental life includes awareness of its own concepts, beliefs, desires, intentions, inferences, actions, and skills. All of these are represented by self-descriptions in a single sort of language, so that the program has access to all of these aspects of itself, and can reason about them in the same terms.