# THE ARCHITECTURE OF A COOPERATIVE RESPONDENT (Dissertation Proposal)

*Brant A. Cheikes*

**MS-CIS-89-13**
**LINC LAB 143**

**Department of Computer and Information Science**
**School of Engineering and Applied Science**
**University of Pennsylvania**
**Philadelphia, PA 19104**

**February 1989**

---

**Acknowledgements:** This research was supported in part by DARPA grant NOOO14-85-K-0018, NSF grants MCS-8219196-CER, IRI84-10413-AO2 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

**Abstract**

If natural language question-answering (NLQA) systems are to be truly effective and useful, they must respond to queries *cooperatively*, recognizing and accommodating in their replies a questioner's goals, plans, and needs. Transcripts of natural dialogue demonstrate that cooperative responses typically combine several communicative acts: a question may be answered, a misconception identified, an alternative course of action described and justified. This project concerns the design of *cooperative response generation systems*, NLQA systems that are able to provide integrated cooperative responses.

Two questions must be answered before a cooperative NLQA system can be built. First, what are the reasoning mechanisms that underlie cooperative response generation? In partial reply, I argue that *plan evaluation* is an important step in the process of selecting a cooperative response, and describe several tests that may usefully be applied to inferred plans. The second question is this: what is an appropriate architecture for cooperative NLQA (CNLQA) systems? I propose a four-level decomposition of the cooperative response generation process and then present a suitable CNLQA system architecture based on the blackboard model of problem solving.

# Contents

# Chapter 1

# Introduction

This report describes proposed research on the design of *cooperative response generation systems*, systems that are able to generate replies accommodating the goals, plans, and needs of their users. Although this report focuses on cooperative behavior in natural language interaction, I will argue later that the principles involved are not limited to natural language processing.

This thesis defends the following claims:

- Evaluation of what is taken to be the questioner's plan is an important step in cooperative response generation;
- The cooperative response generation process can be modeled using a constrained four-panel blackboard architecture.

In addition, this thesis gives evidence for several ancillary claims, including:

- Cooperative response generation may be usefully viewed as a process of choosing a set of cooperative acts to perform in the response;
- Cooperative response generation is *reflective* in that some choices of response content are motivated not by understanding of the question and its purpose but by needs created by other response-content decisions;
- The blackboard architecture is particularly well chosen for cooperative system design on several practical grounds, foremost among them *modifiability*.

This chapter introduces the major claims and sets the research proposal in the context of other research on cooperative interaction.

## 1.1 Responding Cooperatively

Our success in day-to-day affairs depends largely on the cooperation of those with whom we interact. Should I encounter another person before a closed door while my hands are occupied (I am carrying several cases of beer, for example), I expect that person to open the door for me. More often than not, that person will open the door without any prompting. But even if I must request it explicitly, I expect the person to willingly open the door rather than, say, shrug and walk away.

Our expectations about cooperation affect the way we communicate in natural language. When we ask questions, we frequently intend that our questions be taken not at face value but rather as clues for deducing our goals, plans, and needs. To assume that our interlocutor is cooperative

**Q:** I want to talk to Kathy. What is the phone number at the hospital?
**R:** Kathy is not at the hospital any more. She was discharged yesterday and is at home now. Her home phone number is 222-4321.

Figure 1.1: Example from Pollack's dissertation

means to expect him or her to make that effort, to determine the purpose of our question and to respond to that purpose. There are many examples in the literature supporting the claim that consideration of a question's purpose informs the choice of a reply. The most familiar example, adapted from Grice [22], is this: suppose you are approached on the street by someone carrying an (obviously empty) gasoline can who asks, "Where is the nearest gas station?" To reply, "The Texaco two blocks down on your left," in full knowledge that the Texaco station was closed, would be judged unhelpful and inappropriate despite completely and truthfully answering the question. Why? Because the purely truthful response ignores the purpose of the query: it is not about locating the nearest gas station, it is about obtaining gas.

Assuming cooperativeness increases the efficiency of natural language communication by reducing or eliminating the need to ask or explain questions in detail. In cooperative interaction, some of the computational burden is shifted to the respondent. As long as established rules of communication (such as Grice's Cooperative Principle and its attendant maxims) are obeyed, the questioner can rely on the respondent's ability to infer what was left implicit in the query.

Within the natural language research community it has long been understood that natural language interaction often obliges question-answering (NLQA) systems to take the initiative when responding rather than answering queries passively. NLQA systems cannot simply translate input queries into transactions on database or expert systems; instead, they must apply many more complex reasoning mechanisms to the task of deciding how to respond. It has been suggested that cooperative NLQA systems must be able to provide *extended responses* [55, 56], combining such elements as:

- a direct answer;
- information or action that is pertinent to the direct answer;
- information or action pertinent to one or more of the questioner's stated or inferred goals.

Consider the example in Figure 1.1 taken from Pollack's dissertation [47].[1] In that example, R does not give Q the information he requested explicitly. Instead, her extended response (indirectly) explains why Q should not try to call Kathy at the hospital, tells Q where Kathy can be reached, and provides Q with the telephone number. R recognizes Q's goal of talking to Kathy (as opposed to calling the hospital) and accommodates that goal by giving Kathy's home telephone number.

The literature on cooperative communication is extensive. At the University of Pennsylvania alone, several projects have focused on cooperative response behavior, including Kaplan's work on responding when presuppositions fail [31], Mays' work both on responding when queries fail intensionally [35] and on determining competent monitor offers [36], McKeown's TEXT system

[1] In this and all future examples, 'Q' is used to denote the questioner, 'R' the respondent (system). When useful for clarity, feminine pronouns will be used to refer to R and masculine pronouns to refer to Q.

for explaining concepts known to a database system [38], McCoy's system for correcting object-related misconceptions [37], Hirschberg's work on scalar implicatures and their use in avoiding the production of misleading responses [25], and Pollack's plan inference model for recognizing and responding to discrepancies between the system's and the user's beliefs about domain plans and actions [48, 47]. Other explorations of cooperative communication include [1], [6], [8], [16], [18], [28, 27, 29], [39], [49], [55], [56], and [59]. Of course, I have mentioned only a few of the many contributions in this area.

For the most part, however, these efforts have been conducted in isolation; no serious attention has been paid to the task of designing a system or system architecture that could integrate in any general way even a small subset of the kinds of behavior treated by the various researchers. When systems were implemented, their designers focused on developing the reasoning mechanisms and encoding the knowledge needed to produce the particular forms of cooperative behavior under study. The architectures used were tailored to the testbed application.

It is thus an appropriate time to focus on the design of cooperative response generation (CRG) systems. In that endeavor, there are two questions to be answered. The first is this: what are the reasoning mechanisms that underlie cooperative response generation? In partial reply, I argue that *plan evaluation* is an important step in process of selecting a cooperative response, and describe several tests that may usefully be applied to inferred plans. The second question is: what is an appropriate architecture for cooperative NLQA (CNLQA) systems? In answer to this question, I propose a four-level decomposition of the cooperative response generation process and then present a suitable CNLQA system architecture based on the blackboard model of problem solving. In the next section, I will describe the restricted form of CNLQA system that this research assumes.

## 1.2 Cooperative NLQA Systems

A CNLQA system is a NLQA system that is able to perform cooperative response generation. As this thesis is a first pass at exploring the problems of CNLQA system design, I will assume a restricted CNLQA system model in order to make the problem more manageable.

Input to the CNLQA system I envisage may consist of one or more NL utterances. Of those utterances, only one may be a question (request). Those utterances *excluding* the question will be assumed only to express features of Q's domain beliefs, his intended actions, desired goals, or preferences. Thus, inputs like this are allowed:

> **Q:** How late is the Parkvale Deli open? I need to buy some beer for a party tonight.

As transcripts of natural dialogue show, questioners often provide many other kinds of information along with their requests, and occasionally make multiple requests. For example, it is not uncommon for people to describe prior actions they took, the perceived results of those actions and how the results differed from their expectations, and so on, before asking their question. Attempting in this thesis to deal with the range of such input would open up many difficult problems of interpretation, representation, and reasoning; therefore I am adopting a simplified CNLQA system model.

Within this simplified framework, I take the cooperative response generation task to be: given a question (and its accompanying utterances, if any), compute a plan specifying the information to be conveyed in response that the system believes will satisfy—as well as possible—the questioner's needs.

A definition of what best satisfies the questioner's needs in a given situation should be based on a precise and comprehensive theory of cooperative communication. Unfortunately, no such theory

exists. It is a difficult problem: though we know about many potential elements of a cooperative response, such as direct answers and refutations of misconceptions, we cannot enumerate a set of rules that prescribe the complete cooperative response behavior of a respondent to a question. Yet the amount of work done in the area attests to the fact that we have strong intuitions about what distinguishes a cooperative from an uncooperative response.

I will make no attempt here to propose a theory of cooperative communication. Instead, I will take advantage of the precedent set by earlier work and rely on examples to provide an informal definition of cooperative response behavior. In general, I will consider a response cooperative if it is both *informative* and *non-misleading*.

An *informative* response provides correct information that helps the user achieve his goal. When obstacles (used in the sense of Allen [1]) are detected in the user's plan, the system should provide information to eliminate them, or should at least make the user aware of them. The response should address the user's immediate needs. Less immediate needs may be addressed if doing so is deemed useful in some way.

A *non-misleading* response does not license false inferences. In particular, extensional query failure should be handled appropriately (following Kaplan [31] for example) and false implicatures of scale [25] should be prevented.

There are a few other noteworthy simplifications I will assume. In the system I will develop, the output will be limited to a representation of information to be conveyed rather than an actual NL response. This is based on the hypothesis that CRG can be divided into two sequential tasks: first deciding what information to convey in a response, then deciding how to present that information to a user (e.g., what grammatical structures and lexical items to use). For this thesis, I address only the first part, designing a system to produce a representation, in some representation language independent of any particular natural language, of the information to be communicated in its reply.

I consider only single-turn exchanges. That is, the system is given some input utterances and computes a response that does not assume an extended discourse. Discourse phenomena in general will be ignored in this thesis; for example, I won't consider how previous interactions influence the choice of a cooperative response.

Finally, I am limiting my attention to *information-seeking dialogues*—dialogues in which a questioner poses a request for information to a respondent. Among others, advisory dialogues and collaborative-task dialogues, in which system and user work together to solve a problem, are excluded. System output will consist entirely of linguistic rather than physical acts.

Despite these simplifications, the problem remains difficult. Many instances of question-answer exchanges recorded in transcripts of natural dialogue are of the restricted form just described. Furthermore, many of the previously mentioned research efforts are similarly restricted in scope, creating the opportunity to consider how other approaches might be incorporated into the design proposed here.

## 1.3 Cooperative Response Generation

This thesis takes the view that when presented with a question, a cooperative respondent first generates a set of response goals—goals describing the information that the respondent desires to communicate in her response. These goals reflect various cooperative decisions made by the respondent and are produced in reaction to the needs of the questioner as the respondent perceives them. For example, if Q asks R whether some proposition P is true, R may adopt the goal of telling Q whether P is true; Q's act of asking whether P leads R to believe that knowing whether P is one of Q's needs. R's response is then an attempt to satisfy Q's perceived need. Plan inference has

been suggested as a means by which respondents discern their questioners' needs.

NL researchers have identified whole classes of cooperative response goals and attempted to understand the knowledge and reasoning that give rise to them. Allen found that several kinds of cooperative behavior—interpretation of indirect speech acts and sentence fragments and the providing of information not explicitly requested—could be accounted for if the respondent was presumed to perform plan inference (cf. Chapter 2, Section 2.2.2). McCoy showed how to enhance a domain model (adding what she called *perspective*) and reason on it in order to produce text that corrected a user's object-related misconception [37].

The implicit assumption of this research approach is that once we circumscribe the knowledge requirements and reasoning mechanisms for each class of cooperative response goal, we can then put them together into a system that can select appropriate combinations of cooperative response goals. The only problem that will then remain is how to realize those goals in a NL response.

This thesis makes the same assumption of decomposability—which I call the *decomposability hypothesis*—and focuses on the problem of how to select multiple response goals. Though as part of the research I will develop a particular taxonomy of cooperative response goals, I will make no claim that it is the only valid such decomposition. Rather, as I explore in this research the usefulness of the decomposability hypothesis to CRG, I hope to gain some insight into the different ways that the cooperative response reasoning process can be partitioned.

Given the decomposability hypothesis, I suggest that the process of computing a cooperative response can be separated into two phases of decision making: first, individual cooperative response goals are suggested by independent reasoning modules, and second, interactions between goals are analyzed and conflicts resolved. That is, I view CRG as a problem of controlling the operation and interaction of a collection of independent modules that work together to form the system's cooperative response plan. This notion will be expanded upon in Chapter 4.

Blackboard architectures, in which modules called *knowledge sources* work together to solve problems, have been used in a number of artificial intelligence systems [43, 42]. Because I am modeling CRG in terms of independent processing elements, I have chosen to develop a blackboard-based architecture for CNLQA systems. In order to familiarize the reader with the blackboard model of problem-solving, Chapter 2 includes a survey of several of the major contributions to blackboard architecture design.

## 1.4 Plan Inference and Plan Evaluation

*Plan inference* is the process by which one agent A attempts to deduce another agent B's immediate and/or long-term objectives and intended courses of action from observations of B's behavior. Researchers have distinguished *intended* and *keyhole* plan inference. In intended inference, B acts so as to communicate enough details of his plan for an observing A to correctly infer it (or at least the pertinent details). In keyhole inference, B makes no effort to communicate his plan and may in fact be totally unaware that A is watching. Hence the name: A observes B as if through a keyhole and tries to guess what B is doing.

Allen and Pollack, among others, have argued that proper understanding of questions in information-seeking dialogue requires plan inference, and that questioners generally intend that the relevant portions of their plans be recognized [1, 47]. The example in Figure 1.2, taken from transcripts of an electronic interaction between an "expert" and a novice user of the Emacs text editor,[2] illustrates the role of plan inference in collaborative task dialogues. Q's task is to use

[2]The Emacs transcripts were collected as part of a project for a natural language seminar during the spring of 1982 [33]. There were eight dialogues recorded, each taking place between two people, one acting as an expert and

**Q:** Beginning of region—How?
**R:** Type ESC-M. It should say "mark set" at the bottom of the screen.
**Q:** It says "execute-mlisp-line."
**R:** Type ESC and try again.
**Q:** Good. Now end of region.
**R:** You move the cursor down to the end of the region. OK?
**Q:** That's fine, how do I mark, ESC-W?
**R:** No, you don't have to mark the end of the region. Only one extreme needs to be marked with ESC-M.

Figure 1.2: Emacs task dialogue

Emacs to convert a region of text in a computer file to upper case. That involves moving the cursor to the beginning of the region, typing escape-M (the escape key followed by the letter "M") to "set the mark," then moving the cursor to the end of the region and typing the command sequence escape-X case-region-upper (the escape key followed by the letter "X" followed by the character string "case-region-upper"). The dialogue begins after Q has positioned his cursor at the beginning of the region.

In order to interpret the fragment "beginning of region—How?", R must use both her beliefs about Q's current state in the task and her beliefs about Q's objective. Her belief that at the time of the question Q's cursor is positioned at the start of the region enables her to rule out the possibility that Q wants to know how to *move* to the beginning of the region. Her knowledge that Q wants to change the region to uppercase coupled with her knowledge that a step in the "uppercase region" plan involves setting the mark at the region's beginning lets her interpret the question as a request to be informed how to set the mark. The "now end of region" fragment is interpreted similarly: her beliefs about Q's plan and his progress in that plan let R decide that Q now wants to *move* to the end of the region.

*Plan evaluation* is the process of testing an inferred plan for errors due to such things as missing or incorrect domain knowledge or misconceptions about domain objects or actions. With reference to the dialogue of Figure 1.2, I suggest that it is R's evaluation of Q's plan that leads her to point out that the end of the region should not be marked. At the point in the dialogue where Q asks "how do I mark, ESC-W?", R believes that Q's cursor is positioned at the end of the region to be converted to upper case. Ordinarily, R would expect that Q would be intending next to invoke the "case-region-upper" command. Instead, Q has asked (again) how to set the mark. To properly interpret the request, R must first modify her beliefs about Q's plan to include a set-mark action after the move-to-end-of-region action, and then evaluate that plan to discover that she believes it is incorrect, in particular, that she believes the second set-mark action will thwart Q's goal. It is R's knowledge that Q's plan is inappropriate, and specifically her knowledge of *the way* that plan is inappropriate, that leads her to explain that the mark is only set at one end of the desired region.[3]

the other as a novice as they communicated via a computer terminal. I have made some minor changes to increase clarity.

[3]This example is also interesting in that it shows that plan inference and evaluation are ongoing processes. At each turn, R must update her beliefs about Q's progress in his plan and use those beliefs when interpreting Q's utterances.

In Chapter 3, I will discuss in more detail (with examples) the reasons why plan evaluation should be included in the model of cooperative response generation. I will also suggest a couple of approaches to the design of a plan evaluation system. To provide some background on the plan evaluation problem, the next chapter includes a review of several relevant research efforts.

## 1.5 Outline of Proposal

The next chapter surveys the literatures of both blackboard architecture and plan inference and evaluation. From the reading on blackboard systems, the reader can get a sense of the diverse nature of the problems to which the architecture has been successfully applied. The literature on plan evaluation is sparse; much of the review concerns work on plan inference and critiquing, with attention paid to the ways in which various systems test inferred plans for executability or usefulness.

Chapter 3 describes my approach to plan evaluation. Chapter 4 explains the theory behind and the design of a constrained blackboard architecture for the CRG task. It shows by example how a system built according to the proposed architecture could compute cooperative responses. Chapter 5 concludes this report by discussing my plans for carrying out the proposed research.

# Chapter 2

# Survey of Related Research

This chapter reviews published research in two areas: blackboard (and related) system architectures, and plan inference and evaluation as applied to natural language processing.

## 2.1 The Blackboard Model of Problem Solving

The earliest blackboard architecture underlay the HEARSAY-II speech understanding system [14]. In its simplest form, the blackboard architecture has three properties:

1. *All problem-solving state information is maintained in a shared knowledge base called the* **blackboard**. The blackboard is often organized into regions, each representing the state of the solution at a different (problem-specific) level of abstraction. For example, a high level of abstraction might correspond to a general problem solving strategy, while a lower level could translate to a particular set of actions to take.

2. *Elements of the solution are computed and recorded on the blackboard by independent processes called* **knowledge sources**. Each knowledge source (KS) is defined by (1) the conditions under which it can contribute to the problem-solving process, and (2) its problem-solving behavior. KSs act by making changes to the blackboard, and do so independently—without considering the existence or expertise of other KSs. Communication (and coordination) between KSs occurs anonymously and indirectly through changes made to the blackboard.

3. *Problem-solving behavior is opportunistic and incremental.* KSs are self-activating: when a KS determines that its triggering conditions are satisfied, it immediately performs its problem-solving function.[1] Problem solutions are developed incrementally from the combined contributions of the KSs.

Nii points out that the blackboard paradigm provides a set of organizing principles rather than a computational specification [43]. As a result, the designs of different blackboard systems usually vary in implementation according to how the blackboards are structured and how KS activity is controlled. This will become clear in the next several subsections, in which I will review several representative blackboard systems.

[1]The basic blackboard architecture completely sidesteps the issue of control. In real systems, a *scheduler* of some sort controls KS activation and execution.

### 2.1.1 The HEARSAY-II speech understanding system

The HEARSAY-II system (henceforth HEARSAY), was developed between 1971 and 1976 as part of a DARPA-sponsored five-year speech understanding program [14, 42]. Its task was the real-time recognition of connected English speech taken from a vocabulary of about a thousand words.

HEARSAY's blackboard was arranged as a hierarchy that fit the levels of analysis needed for the speech understanding task: *parametric*, *segmental*, *phonetic*, *surface phonemic*, *syllabic*, *lexical*, *phrasal*, and *semantic*. Each blackboard change took place within one of these levels. A KS triggered by a blackboard change would typically generate relevant hypotheses at the next higher level of analysis. For example, a KS noticing the appearance of a pair of adjacent syllables at the syllabic level might generate hypotheses at the lexical level corresponding to the possible words that the syllables might comprise.

KS activation was controlled through the combined work of a *blackboard monitor* and a *scheduler*. The monitor kept track of all changes made to the blackboard. After each change, the monitor used its knowledge of the KSs[2] to select those which could potentially be executed as a result of the change. Pointers to KSs ready for execution were maintained on a list of *invoked knowledge sources*. On each system cycle, the scheduler computed a priority (using experimentally derived heuristics) for each KS on that list and executed the one with the highest rating. Because the HEARSAY scheduler used knowledge about the KSs to determine priorities, it is often described as a *sophisticated scheduler*.

After some experimentation, it became evident that a method was needed to minimize the number of KS executions needed to make progress toward the solution, the so-called *focus of attention* problem. Although "sophisticated," HEARSAY's scheduler still did not have enough information about the KSs in order to most effectively apply them during problem solving.

### 2.1.2 HASP: a system for ocean surveillance

The HASP system (later renamed SIAP) was designed to analyze the movements of surface ships and submarines within an ocean region. It did this by interpreting continuous sonar signal input passively collected by an array of hydrophones [43, 42, 46]. The system had multiple input streams: besides receiving continuous acoustic signals from each hydrophone, HASP was also provided with intelligence reports that detailed the movements of friendly and hostile ships in the region, including commercial shipping activity.

At any given time, the HASP blackboard recorded the system's best evaluation of the situation. For this reason, the blackboard was referred to as the *current best hypothesis* (CBH). The CBH was partitioned into a level-of-analysis hierarchy that fit the signal interpretation task: *input segments*, *lines*, *harmonic sets*, *acoustic sources*, *platforms*,[3] and *fleet levels*. The acoustic input appeared on the segments level, intelligence reports on either the fleets or platforms level depending on the nature of the report. The intelligence reports were used to help constrain the possible interpretations assigned to the acoustic data.

HASP extended the blackboard model (in a limited way) to incorporate control-level decision making. Four categories of system events were defined: *clock* events provided a means to cause rules to execute at particular times, *problem* events concerned difficulties encountered by the KSs, *expectation* events involved situations expected to occur in the future, and *blackboard* events dealt with the activation of problem-solving KSs. Control consisted of a *strategy* KS and a set of *event-*

[2] A certain amount of declarative knowledge about each KS, including its activation conditions and typical behavior, was stored in the system and available for use by both the monitor and scheduler.

[3] A "platform" is intelligence jargon meaning a ship.

*management* KSs. Based on priorities in its database, the strategy KS chose the event category to process next. The corresponding event-management KS was then called to select and execute the particular event in the chosen category. Thus limited control decisions could be made about problems on which to next focus the system's attention. Though called knowledge sources, control KSs operated outside the blackboard structure, requiring control knowledge to be expressed and processed differently from the domain problem-solving knowledge. This situation is remedied by BB1, discussed next.

### 2.1.3 Blackboard control architecture

The success of the HEARSAY and HASP projects popularized the blackboard paradigm, and a number of other systems were built along the same lines (some of which are summarized in Section 2.1.5). As more researchers began experimenting with blackboard architectures and applied them to different application domains, the deficiencies of the model became apparent. For example, Corkill *et al.* argued that the HEARSAY architecture lacks the ability to perform goal-directed reasoning: HEARSAY-style systems were able to plan their problem-solving activities in only the most rudimentary ways, usually by manipulating the assignment of priorities to KS activation records [11]. Corkill proposed to extend the model by adding a second blackboard, using one for the usual task-level problem solving and the other (called the *goal blackboard*) for reasoning about the system's problem-solving goals. He argued that the addition of the goal blackboard enabled goal-directed reasoning.

This natural evolution of blackboard architecture was considerably revised and extended by Hayes-Roth, who dubbed the result *blackboard control architecture* [23]. She designed a domain-independent framework, called BB1, for implementing systems with blackboard control architectures. Using BB1, she implemented the OPM system which performs multiple-task planning.

Just as Corkill proposed, the BB1 architecture extends the blackboard model of problem solving by distinguishing separate blackboard panels for domain and control level reasoning. The *control problem* is that of selecting, from a set of several possible actions, the next one to take to advance the solution state. The principal contributions of Hayes-Roth's work are her detailed characterization of intelligent problem-solving control and her elaboration of an architecture that provides a uniform, adaptive mechanism for scheduling the operation of both domain and control KSs.

#### Intelligent problem solvers

Hayes-Roth argues that the control problem must be explicitly and dynamically solved during the course of problem solving, and that intelligent problem solvers must (p. 252):[4]

1. Make explicit control decisions that solve the control problem.
2. Decide what actions to perform by reconciling independent decisions about what actions are desirable and what actions are feasible.
3. Adopt variable grain-size control heuristics.
4. Adopt control heuristics that focus on whatever action attributes are useful in the current problem-solving situation.
5. Adopt, retain, and discard individual control heuristics in response to dynamic problem-solving situations.

[4] Unless otherwise noted, all page numbers cited in this section are references into [23].

6. Decide how to integrate multiple control heuristics of varying importance.

7. Dynamically plan strategic sequences of actions.

8. Reason about the relative priorities of domain and control actions.

These goals are elaborated in the next several paragraphs.

**Explicitly solving the control problem** Hayes-Roth argues that control decisions must be made explicitly, rather than embedded in the scheduling algorithm used to guide knowledge source execution: "Intelligent problem-solving entails explicitly solving the control problem—*deciding* which actions to perform and when to perform them—as well as performing selected actions" (p. 255).

**Reconciling decisions about desirable vs. feasible actions** At each stage in the problem-solving process, a system must reconcile decisions it has made about what actions it should perform next and what next actions are actually possible. Given a control heuristic to favor actions having high priority, a high priority action might be selected for execution. As another example, a system might desire to perform an action having a certain property and then discover that none of the available next actions have that property. One possible resolution would be to adopt a control strategy favoring actions that *enable* the performance of the currently desired action. Thus both the desirability and feasibility of potential actions guide the system's control decision-making.

**Variable grain-size control heuristics** Control heuristics come in different grain sizes. Some, for example, dictate the application of certain classes of actions, while others dictate the use of particular actions. At each control decision point in the problem-solving process, an intelligent problem solver adopts control heuristics at whatever grain size is most appropriate.

**Focusing on useful action attributes** Attributes of potential actions may serve as useful bases for control decisions. For example, in situations where potential actions may differ in credibility, a control decision might be made to focus on tasks that produce reliable results. When problem-solving time is limited, control may focus on actions that can be performed quickly.

**Dynamically adopt, retain, and discard control heuristics** Intelligent problem solvers respond dynamically to changes in the state of the problem-solving process. As new needs arise, appropriate control heuristics are selected and obsolete ones abandoned.

**Integrating multiple control heuristics** At any stage during problem solving, several control heuristics may simultaneously apply. The system must be able to integrate all pending heuristics when deciding which action to perform next. This may mean obeying the highest priority heuristic over the others, or choosing actions that satisfy subsets of the heuristics.

**Planning strategic action sequences** Greater power may be derived by planning sequences of actions as opposed to selecting individual ones. For example, a system might choose to focus on developing a general plan for one complex subproblem of the task at hand, and then refine that plan down to the individual actions. In that way, control heuristics would favor *sets* of actions that could be taken to advance the solution. Planned action sequences should be interruptable during execution in response to changes in the solution state.

**Relative priorities of domain and control actions** An intelligent problem solver selects actions to solve the control problem in tandem with selecting actions to advance the solution of the domain problem. "Because there is no *a priori* reason to presume that either type of action is more productive on any particular problem-solving cycle, the system reasons about the relative priorities of domain and control actions just as it reasons about other control issues" (p. 259–260). So a problem solver may need to choose between, say, performing a domain action that will connect two solution islands, and adopting a new control heuristic to favor actions which execute quickly.

### The blackboard control architecture

The BB1 blackboard control architecture follows the same basic control cycle as other blackboard models:

1. Enumerate the set of KSs whose triggering conditions and preconditions are met;
2. Select one KS for execution;
3. Execute the selected KS.

As indicated, KSs have both *triggering conditions* and *preconditions*. Triggering conditions are event-based predicates describing the kinds of blackboard events (modifications) that a KS reacts to; preconditions are state-based predicates that define the system state necessary for the KS to execute. When a KS is triggered, a *knowledge source activation record* (KSAR) is created for it, containing such information as the name of the triggered KS, its preconditions, values characterizing its credibility and efficiency, and its expected behavior. KSARs are used by the control KSs to make scheduling decisions.

BB1 defines explicit domain and control blackboards and distinguishes domain and control KSs. The domain blackboard is structured according to the needs of the problem whereas the structure of the control blackboard is defined by the architecture. Domain KSs operate using the same methods used in standard blackboard systems: they read and modify the domain blackboard. Control KSs operate on the control blackboard, making decisions that affect the scheduling of both domain and control KSs. Several domain-independent control KSs are defined by the architecture, but domain-specific control KSs are also allowed. The next section reviews the BB1 approach to control reasoning in more detail.

### The control blackboard

Control KSs "dynamically compose a prescriptive control plan out of modular control heuristics" (p. 268). The blackboard that they operate on represents six categories of control decisions: *problem*, *strategy*, *focus*, *policy*, *to-do set*, and *chosen action*. These are reviewed in the following paragraphs.

**Problem level** "A single Problem decision represents the problem a system has decided to solve and guides the entire problem-solving episode" (p. 270). Problem decisions initiate processing by triggering one or more domain or control KSs. The problem decision includes predicates defining an acceptable solution; processing ends when those criteria are satisfied. If necessary, KSs are allowed to modify problem decisions. For example, if time is limited, a control KS may update the problem decision and relax some of the solution-acceptance criteria in order to allow a solution that is less than ideal.

**Strategy level** "Strategy decisions establish general sequential plans for problem solving" (p. 272). Strategies declare lists of foci for the focus level (described next). KSAR scheduling decisions are directly affected by focus decisions; strategies indirectly influence KSAR scheduling by mapping out sequences of focus decisions. Successive refinement, in which a plan of action is developed at successively more detailed levels, is one example of a strategy decision. Though a single strategy decision usually operates throughout the problem-solving episode, control KSs may modify or replace strategy decisions in response to dynamically changing conditions.

**Focus level** Focus decisions directly affect KSAR scheduling by establishing rating criteria. The rating criteria cause KSARs with certain attributes and values to be favored by the scheduler. The effect of focus decisions is to set "local problem-solving objectives" (p. 273). For example, a focus decision could be made to favor KSARs whose problem-solving activity takes place at the syllabic level of the domain blackboard (to recall the structure of HEARSAY's blackboard). The result of such a focus decision would lead the scheduler to rate more highly those KSARs that hypothesize syllables from phonemes. Focus decisions direct the order in which invocable KSs are executed, thus guiding the system's overall problem-solving behavior.

**Policy level** Policy decisions are similar to focus decisions in that they specify global criteria for scheduling KS activity. They differ in terms of longevity: focus decisions are typically short-lived (existing until the focused goals are satisfied) while policy decisions usually last from the time they are created until problem solving is completed. An example of a policy decision would be one that favors KSs whose results are reliable or one favoring efficient KSs. Of course, the architecture still allows control KSs to add, remove, or modify extant policy decisions.

**To-do set level** The to-do set level records the list of pending KSARs on each system cycle, and is structured into two lists, the *triggered list* and the *invocable list.* When the triggering conditions of a KS are satisfied, a KSAR is created for it and put on the triggered list. When the KSAR's preconditions are satisfied, it is moved from the triggered list to the invocable list. KSARs on the invocable list can be scheduled for execution. Because invocable KSARs are not executed immediately, it is possible for one or more of their preconditions to subsequently become false (due to the problem-solving activity of other KSs). When this happens, the KSAR is moved back to the triggered list. Such shifting between the triggered and invocable lists happens freely. Once triggered, however, a KS can never become "untriggered" and removed from both lists.

**Chosen action level** Chosen action decisions record the scheduler's choice of the next KS to execute.

### Generic control KSs

BB1's intelligent problem-solving behavior is implemented by three generic control KSs: *Update-To-Do-Set*, *Choose-KSAR*, and *Interpret-KSAR.* These KSs are special in that they implement the system's basic operating cycle, and thus fall outside it—they do not compete with the other domain and control KSs for execution.

**Update-To-Do-Set** This KS is invoked after the completion of each designated chosen action. Its job is two-fold: first it must identify all domain and control KSs whose triggering conditions are met, create KSARs for them, and place those KSARs on the triggered list of the to-do set level of

the control blackboard. Second, it moves all triggered KSARs whose preconditions are satisfied to the invocable list and all KSARs on the invocable list whose preconditions are not satisfied to the triggered list. It also performs a number of bookkeeping chores, including computing ratings for each KSAR using all relevant Focus and Policy decisions.

**Choose-KSAR** Choose-KSAR uses all scheduling rules in effect to select for execution a KSAR from the to-do set. These scheduling rules exist in the form of Focus and Policy decisions—thus focus and policy decisions specify not only how KSARs are rated for execution but also how those ratings are used to select a KSAR to be run. Choose-KSAR records the selected KSAR on the chosen action level of the control blackboard.

**Interpret-KSAR** This KS simply causes the chosen action to be executed and performs all the necessary bookkeeping.

#### BB1 in summary

BB1 significantly enhances the HEARSAY-style blackboard model of problem solving, achieving all eight behavioral goals that Hayes-Roth sets for intelligent problem solvers. Though the BB1 kernel is still a simple control loop, there are no built-in scheduling algorithms. The BB1 environment provides a means for creating problem solvers that can dynamically alter their problem-solving behavior in response to changing conditions. Domain and control reasoning is represented and controlled in a uniform manner. Because KSs act independently, the overall architecture is highly modular, a feature of particular value to developers of experimental systems. The main problem with BB1 appears to be the potential loss in system performance due to the overhead of control reasoning, a problem that has been studied with encouraging results.

#### Other work on BB1

The BB1 architecture has been used successfully in a number of applications including the PROTEAN system which models the structure of protein molecules [30]. As part of that work, some straightforward extensions were made to BB1 to support goal-directed reasoning.

Garvey *et al.* have conducted some experiments to determine whether the added costs of performing control reasoning are outweighed by the potential efficiencies gained by doing so [19]. Their experiments, conducted with the PROTEAN system, showed that increased control reasoning led to reductions in total problem-solving time by reducing the number of actions performed, especially those calls to expensive domain-dependent numerical calculation routines. In another set of experiments, Garvey and Hayes-Roth compared the benefits of implicit versus explicit control architectures, using the PROTEAN system as a test bed [20]. They concluded that while procedural (implicit) architecture generally outperformed explicit control architecture, the BB1 approach had several other benefits that justified the performance penalty, including clarity and uniformity of representation for task and control level reasoning. Further discussion of the (present and future) benefits to developers using the BB1 approach can be found in Hewett [24], while Schulman describes how a general explanation facility can be easily integrated into BB1 applications [52].

### 2.1.4 UCEgo: kernel of an "intelligent agent"

UCEgo, part of the Unix Consultant (UC) project [59] under development at the University of California at Berkeley, coordinates UC's decision-making leading to the production of a cooperative response [7]. Although UCEgo was not designed to be a blackboard system, its architecture

nevertheless has some of the flavor of the blackboard approach. Before discussing UCEgo itself, a brief overview of UC is in order.

UC is a NLQA system that answers questions about the Unix[5] operating system. In UC, natural language utterances are processed first by ALANA, a language analyzer which constructs a meaning representation of the input in the KODIAK semantic network representation [58]. The resulting KODIAK structure is then passed through a *concretion* mechanism which performs various kinds of inference and modifies the network accordingly [57]. The revised KODIAK network is next passed to PAGAN, UC's Plan and Goal Analyzer. PAGAN augments the KODIAK network with information describing the system's beliefs about the user's high-level and immediate goals. At this point, UCEgo takes control, making decisions about how UC should respond. According to Chin, UCEgo transforms UC into an "intelligent agent" capable of detecting its own goals and forming and executing plans to achieve those goals. UCEgo works in conjunction with KNOME, the user modeling component, and UCPlanner, a domain planner that generates plans for activities in Unix. The output of UCEgo is a representation (again in KODIAK) of a set of speech acts. These speech acts are turned over to UCExpress, which performs more KODIAK-to-KODIAK transformations, this time refining the output of UCEgo to a format that can be expressed clearly and coherently. The output of UCExpress is then fed to the last element of UC's pipe, UCGen, a tactical level natural language text generator, which produces the NL response.

The operational core of UCEgo is a mechanism called an *if-detected daemon.* If-detected daemons are analogous to the knowledge sources of the blackboard model. Each daemon embodies a condition-action rule: the condition part (called the *detection net*) consists of a hunk of KODIAK network to be matched against the KODIAK representation of the query, and the action part (the *addition net*) describes additions to be made to that representation. In effect, the KODIAK hunk provided to UCEgo is placed on an anonymous blackboard. The if-detected daemons then act as autonomous, self-activating KSs: when the detection net matches part of the network on the blackboard, the addition net is immediately instantiated (executed) and added to the blackboard. Scheduling in UCEgo is a matter of sequentializing the activation of the if-detected daemons; there is no prioritization or other form of HEARSAY-style sophisticated scheduling. In principle, if-detected daemons can all be activated in parallel, so there is no need for an explicit scheduler.

If-detected daemons are used to perform all UCEgo's functions. For example, they are used to assert appropriate goals for UC, at all levels from high ("be cooperative") to low ("apologize for lack of knowledge"), and to resolve conflicts between goals as needed. This approach suggests an interesting comparison between UCEgo's architecture and BB1. In UCEgo, there is no distinction between domain and control level reasoning. A single "blackboard" records both control decisions (assertions about goals UC should adopt) and domain actions (assertions about appropriate response speech acts); a single monolithic KODIAK network structure represents not only the semantic content of the user's input, but also UC's beliefs about the user's plans, UC's communicative goals, and UC's intended output. BB1, on the other hand, distinguishes separate blackboards for domain and control decision-making and organizes those blackboards as appropriate to the needs of the task. BB1 depends on the representation being used only to the extent that all KSs communicate in a mutually understandable language.

### 2.1.5 Related studies

The blackboard model of problem solving is now well established, and has led to a number of follow-on works, several of which are grouped and summarized in this section.

[5] Unix is a trademark of AT&T.

**Reasoning in uncertain domains** A modification to the HEARSAY-II architecture that supports reasoning over abstractions of plans is proposed by Durfee and Lesser [12], a revision intended to improve the system's ability to cope with reasoning in highly uncertain domains. According to their method, a system first develops an abstract view of the current problem-solving situation. That abstract view is then used to predict the future significance and cost of alternative problem-solving activity, with plan generation, monitoring, and repair interleaved with plan execution.

**Abstract blackboards** Corkill *et al.* have been developing a system that provides abstract views of blackboard objects and operations [10, 9]. Their system, called GBB, provides a language for describing the structure of a blackboard and its implementation for a particular application in terms of a set of abstract objects and operations. GBB then generates a blackboard database kernel tailored to that application. They argue that the GBB approach makes it possible to modify the implementation of the blackboard database, insertion/retrieval strategies, and the representation of blackboard objects without having to in turn modify domain or control knowledge sources.

**Distributed processing** Ensor and Gabbe extend the blackboard paradigm to *transactional blackboards*, in order to exploit the potential for distributed processing in blackboard systems [13]. Their extension models blackboard accesses as database transactions, and provides a technique for managing asynchronous requests to read data from or write data to a central blackboard.

The CAGE and POLIGON systems [44, 45] are two other attempts at adding distributed processing power to the blackboard model of problem solving. In CAGE, application of parallel processing is controlled by the system designer; system primitives are used to specify concurrency of rule firing and KS activity. POLIGON takes the opposite approach: the KSs themselves decide when to run; there is no global control. Both systems have their problems and are still very much in development.

**HEARSAY-III** HEARSAY-III extends HEARSAY-II to a domain-independent framework for knowledge-based expert systems [4, 15]. It provides facilities for reasoning about partial solutions and describing the application of domain-dependent consistency checks during problem solving. Moving in the same direction as BB1, HEARSAY-III segments the blackboard into domain and scheduling panels. The scheduling KSs in HEARSAY-III are much more limited than in BB1—modifications made to the scheduling blackboard affect the way domain KSs are selected. This is typically done by adjusting assigned priorities or recording small amounts of meta-information on the state of the domain blackboard.

### 2.1.6 Summary

The blackboard model of problem solving is now a well-established AI methodology. Since the original HEARSAY project, blackboard systems have been applied to many different problems in diverse domains. The BB1 framework, the focus of a significant amount of work at Stanford University's Knowledge Systems Laboratory, has been studied with particular care. The blackboard paradigm has been widely noted for its modularity, its adaptability to a variety of reasoning and knowledge application strategies, and the ease with which blackboard systems can be developed. Because the blackboard approach allows integration and control of multiple knowledge sources, it appears to be a promising foundation upon which to build systems that will need to combine many different reasoning methods during the construction of cooperative natural language responses.

## 2.2 Evaluating Plans

Unlike the situation for blackboard systems, work on plan evaluation is scarce, usually implicit or otherwise embedded in larger works. Therefore, the focus of this survey is on the kinds of tests that have been performed on plans. From this, I will be able to draw some conclusions about the essential functions of a comprehensive plan evaluator.

### 2.2.1 NOAH: constructive critics aid plan synthesis

Sacerdoti's planning system NOAH applied procedures called *critics* at each stage of plan expansion [51]. "The constructive critics of NOAH constitute that portion of the system that concerns itself with the interactions among individual actions" (p. 29). The critics are particularly useful because NOAH performs *hierarchical planning*: the system first computes a "rough draft" of the plan—omitting many details—then iteratively expands it to whatever level of detail is deemed necessary. Expansions are based solely on local considerations, so the process of expansion may introduce conflicts only detectable from a global viewpoint.

After each plan expansion step, various domain-independent and domain-dependent critics are applied to the *procedural net* (cf. [51, p. 5–23]) representation of the plan. Each critic tests for a specific problem and adjusts the plan as needed. The critics are *constructive* in that they not only detect errors in plans but also make the necessary repairs. The five domain-independent critics that Sacerdoti discusses are: "resolve conflicts," "use existing objects," "eliminate redundant preconditions," "resolve double cross," and "optimize disjuncts." I review each of these briefly in turn.

**Resolve Conflicts critic** This critic examines the branches of conjuncts to be executed in parallel, looking for situations in which an action in one conjunct falsifies one of the preconditions of an action in another. The critic resolves the conflict by introducing an ordering constraint ensuring that actions are performed before their preconditions are denied.

**Use Existing Objects critic** As rough plans are developed, objects to which actions are to be applied must be specified. NOAH avoids binding objects to action variables unless the choice is clear. In those cases where an obvious best choice does not exist, Use Existing Objects may be able to instantiate the action variable with an object found elsewhere in the plan.

**Eliminate Redundant Preconditions critic** After some amount of action by other critics, the plan may be left in a state that contains redundant preconditions. That is, at different points in the plan, actions may be specified intending to satisfy a precondition that will already hold, as a result of earlier action. The Eliminate Redundant Preconditions critic prunes away these unnecessary steps.

**Resolve Double Cross critic** A special problem exists when each of two actions act to deny one of the other's preconditions. In such a case, a simple linear ordering of action execution is insufficient to resolve the conflict (so Resolve Conflicts is of no help here). Resolve Double Cross tries several strategies, including adding extra steps to the plan, to eliminate this source of conflict.

**Optimize Disjuncts critic** Disjuncts in plans represent alternate courses of action; for example, to paint a ceiling, one may need to acquire a ladder or a chair. During planning, each choice must be expanded until one disjunct proves superior (assuming this always occurs). Optimize Disjuncts

examines disjuncts in plans and uses a (crude) metric to rate the choices and eliminate all but the best option.

**Domain-specific critics** Sacerdoti notes that domain-specific critics may be useful as well. For example, in plans that require the use of tools it may be worthwhile to optimize trips to the toolchest, collecting several tools at once, rather than making a trip each time a new tool is needed.

#### Summary

Though the critics were designed to aid plan synthesis, the kinds of conflicts that the critics detect may also exist in the ill-formed plans of NLQA system users. Tests such as those performed by the Resolve Conflicts and Resolve Double-Cross critics will certainly be of use in a plan evaluation component.

### 2.2.2 Allen's plan inference system

In [1], Allen describes a model of plan inference for NL understanding intended to account for (1) how people choose to provide more information than they were explicitly requested for, (2) how people interpret sentence fragments, and (3) how people understand indirect speech acts. While the work is concerned only with the inference problem and the system is limited to producing valid (executable) plans, there still are details relevant to plan evaluation.

Allen suggests that *obstacle detection* is the process linking the three aspects of cooperative behavior listed above. He defines an *obstacle* as a situation that prevents an actor from achieving a goal:

> The major claim of this paper is that many instances of helpful behavior arise because the observing agent recognizes an obstacle in the other agent's plan, and acts to remove the obstacle. [1, p. 108]

The *recognition* of obstacles constitutes what I call plan evaluation. Allen's system focuses on obstacles that arise from lack of knowledge.

Two kinds of obstacles are distinguished: *explicit* and *implicit*. Explicit obstacles are those that are mentioned in the input utterances; for example, given a request for information, the lack of knowledge of the requested information is an explicit obstacle. Other obstacles may be implicit, detectable only by inspection of the inferred plan—given a request for a train's time of departure, lack of knowledge of its departure gate is an implicit obstacle.

Details of the plan inference model are not germane to the topic at hand; in a nutshell, Allen presents a set of domain-independent plausible inference rules and an algorithm for controlling search through a set of expanding *partial plans*. It is the heuristics used during the search that are of interest here.

#### Search control and rating heuristics

A *partial plan* comprises two parts: the *alternative*, "constructed bottom-up using the plan inference rules from the observed action" (p. 126), and the *expectation*, "constructed top-down using the plan construction rules from an expected goal" (p. 126). Partial plans are rated using a set of heuristics involving only domain-independent relations between actions and action bodies, preconditions, and effects. The ratings are used to select the next partial plan to focus the search on. The first two heuristics are:

**H1** Decrease the rating of a partial plan if it contains an action whose preconditions are false at the time the action starts executing.

**H2** Decrease the rating of a partial plan if it contains a pending or executing action whose effects are true at the time the action commences.

In practice, these heuristics prevented the system from inferring unexecutable (H1) or inefficient (H2) plans. Pollack (cf. Section 2.2.3) has pointed out that actors often form unexecutable plans and so cooperative agents must be able to infer plans that contain errors. Heuristics H1 and H2 suggest that plan evaluation must include checking actions both for unsatisfied preconditions and "pre-satisfied" effects. A situation requiring the latter test is exemplified in this exchange:

**Q:** What is the combination to the lock on this door?
**R:** The door is unlocked.

R infers that Q believes knowing the combination is needed to be able to open the door; that explains Q's request. R should infer a plan for Q that includes an UNLOCK action, such that an effect of UNLOCKing a door would be to cause that door to become UNLOCKED. Plan evaluation should mark the UNLOCK action as unneeded since the door is UNLOCKED at the outset.

Heuristic H3 favors plans whose goals are compatible with the observed action:

**H3** Increase the rating of a partial plan if it contains descriptions of objects and relations in its alternative that are unifiable with objects and relations in its expectation.

This seems to be a reasonable heuristic, since it says nothing about the validity of the descriptions. Of course, plan evaluation must be able to detect improper descriptions, as this example from Pollack [47] suggests:

**Q:** How do I set a file's access permissions to faculty read only?

A problem arises when "faculty read-only" is not a valid file permission in the domain. This example is especially pertinent to heuristic H4:

**H4** Increase the rating of a partial plan if the referent of one of its descriptions is uniquely identified. Decrease the rating if it contains a description that does not appear to have a possible referent.

Reference failures crop up all the time in discourse, so plans containing them *must* be inferable and the failures themselves identified through plan evaluation.

The last two heuristics are:

**H5** Increase the rating of a partial plan if an intersection is found between its alternative and expectation, i.e., they contain the same action or goal.

**H6** Increase the rating of a partial plan each time an inference rule is applied.

Neither heuristic appears to suggest much in the way of plan evaluation. They are included here for completeness.

### Summary

It is interesting to note that Allen's system incorporates several kinds of plan evaluation into the inference process itself. This was useful as it served to greatly narrow the search space. Clearly, if cooperative respondents must be able to infer invalid plans they will inevitably face a larger search space.

### 2.2.3 Inferring invalid domain plans

In her doctoral dissertation, Martha Pollack argued that an agent's ability to infer another agent's invalid domain plans is essential to cooperative response behavior [47]. She defined these assumptions underlying then-accepted plan inference models:

**Principle of Parsimony** The inferring agent has no explicit knowledge of invalid domain information *which it knows to be invalid.*[6]

**Correct Knowledge Assumption** All of the inferring agent's knowledge of the domain is valid.

**Closed World Assumption** All valid domain knowledge is known to the inferring agent.

**Valid Plan Assumption** The actor's plan is valid.

**Appropriate Query Assumption** The query being analyzed accurately requests information that the actor needs to achieve his goal.

Pollack's goal was to develop an inference model that abandoned the last two of these assumptions, to "develop methods of reasoning from valid information to determine novel invalid plans" (p. 24).

Pollack distinguished two views of plans, the *data structure* view and the *mental phenomenon* view, taking the latter for the basis of her model. In the data structure view, plans are encoded as data structures representing goals and actions that are performed in particular sequences under particular conditions to achieve some result. The mental phenomenon view represents plans as configurations of beliefs and intentions held by an agent. The bulk of Pollack's thesis was spent developing a formalism for representing and reasoning about plans as mental phenomena.

Of all the plan inference research reviewed in this section, Pollack's is the only one that addresses plan evaluation explicitly and in some detail. In order to understand that discussion, a summary of her formalism is necessary.

#### Actions and generation

Pollack's formalism builds on the notion of *generation* developed by Alvin Goldman [21]. The action $\alpha$ can be said to *generate* the action $\beta$ if by doing $\alpha$, the action $\beta$ is done.[7] So flipping a light switch could be said to generate turning on the light *under the right conditions.* The generation relation only holds between two act-types if the conditions at the performance time are right: "generation is not a relation between act-types *per se*" (p. 47).

Actions are represented as triples of *act-type*, *agent*, and *time.* The act-type denotes a generic action, such as flipping a light switch or depressing a key on a computer keyboard, the agent indicates who performed the action, and the time represents the performance time of the action.[8]

The CGEN($\alpha$,$\beta$,C) relation—for *conditionally generates*—holds when action $\alpha$ generates $\beta$ under the set of conditions C, and all conditions in C are satisfied at the time of performance of $\alpha$.

#### Plans and explanatory plans

Pollack defines *simple plans* as plans consisting entirely of actions related by generation. Informally, an agent G has a simple plan involving acts $\alpha_1$ to $\alpha_n$ if:[9]

[6] Emphasis mine.

[7] This is an oversimplification, but adequate for the purposes of the current discussion.

[8] Allen's interval-based temporal logic [2] is used to represent occurrences of actions.

[9] A precise definition can be found on and about p. 68 of the dissertation.

1. G believes he can execute each $\alpha_i$,
2. G believes, for each $i$, that $\alpha_i$ generates $\alpha_{i+1}$.
3. G intends to perform each $\alpha_i$,
4. G intends, for each $i$, to do $\alpha_{i+1}$ by doing $\alpha_i$.

As we can see, this definition presents simple plans as configurations of beliefs held by an agent.

Given a query, which Pollack restricts to be of the form *how do I do $\alpha$ in order to do $\beta$*, plan inference becomes a process of finding a plan—called an *explanatory plan* or *eplan*—that represents the questioner's beliefs relating the actions $\alpha$ and $\beta$. That is, a respondent R attempts to ascribe a set of beliefs to a questioner Q that represents Q's simple plan to do $\beta$ by doing $\alpha$. The crucial element of this model is that it allows R to attribute to Q simple plans that may be invalid in a number of ways.

### Invalidities in simple plans

Pollack identifies three general categories of invalidities in simple plans: *unexecutability*, *ill-formedness*, and *incoherence*. It is important to note, however, that judgments of validity may differ between questioner and respondent. From Pollack:

> The correctness of the actor's beliefs thus determines the validity of his plan: if all the beliefs that are part of his plan are correct, then all the intentions in it are realizable, and the plan is valid. Validity in this absolute sense, however, is not what is of primary concern in modeling plan inference. What is important there is rather the inferring agent's *judgment* of whether the actor's plan is valid. This judgment depends on whether the inferring agent believes to be correct the beliefs that she ascribes to the actor. So, an inferring agent's judgment of (in)validity differs from absolute (in)validity in two ways: first, the beliefs being judged are those that the inferring agent ascribes to the actor by virtue of believing he has some plan—they may not be beliefs the actor indeed has; and second, the judgment of these beliefs is from the perspective of the inferring agent—she will judge to be incorrect beliefs that are incompatible with her own beliefs. (p. 75)

**Unexecutable acts** The inferring agent R judges an act in the actor Q's plan to be *unexecutable* if she believes that Q cannot perform the act at the desired time. This may be the case for either of two reasons: first, R may believe that such an act is simply not valid in the domain, as illustrated by Q wanting to set a file's access permissions to "faculty read-only" when R believes that such a permission does not exist. In the second case, Q may not be in the *standard conditions* with respect to the act: some external forces prevent Q from performing the act at the desired time. Q can start his car by turning the key in the ignition as long as there's gas in the gas tank, no potato in the tailpipe, and so on.

**Ill-formed plans** A plan is *ill-formed* if a pair of actions $\alpha_i$ and $\alpha_{i+1}$ in the plan fail to form a generational pair—that is, R judges that $\alpha_i$ will not generate $\alpha_{i+1}$ at the desired performance time. This may occur for several reasons: R may simply believe that $\alpha_i$ *never* generates $\alpha_{i+1}$, or R may believe that one or more of the generation-enabling conditions (one of the C in the CGEN relation) will not be satisfied at the performance time of $\alpha_i$.

**Incoherent plans** R deems a plan *incoherent* if she cannot find a set of actions related by generation that link the queried act $\alpha$ to the goal act $\beta$. Incoherence differs from both unexecutability and ill-formedness in that it represents R's inability to find an eplan underlying Q's query. Note that R may judge Q's plan incoherent when she simply lacks the knowledge that makes the plan coherent.

### Pollack on plan evaluation

Plan evaluation figures prominently both in Pollack's theory of question answering and in the implementation of her plan inference model, a system called SPIRIT. Plan inference is the process of finding the eplan underlying the question, evaluation the process of determining whether that eplan is well formed and comprises executable acts. Evaluation precedes response generation:

> ...the result of the plan inference and evaluation processes can be viewed as an annotated eplan, where the annotations mark any ill-formed portions of the eplan, along with the source of their ill-formedness. [...] The annotated eplan is input to the response generation process. (p. 103)

Pollack also argues for the necessity of plan *synthesis*: having discovered Q's plan invalid (either ill-formed or containing one or more unexecutable acts), R may attempt to synthesize a new plan that achieves Q's stated goal. In Pollack's system architecture, the response generator may receive a variety of information from earlier processing, including the inferred eplan (with unexecutable acts and failed generational pairs marked) and any synthesized plans.

### Summary

Pollack presents a strong argument for viewing plans as mental phenomena, and her plan inference formalism—by allowing the inference of certain kinds of invalid plans—greatly extends the range of possible cooperative response behavior. The work is chiefly limited by its inability to handle plans with actions related by *enablement*.

In any reasonably-sized domain where lots of goals and actions are possible, the number of interesting simple plans is small compared to the number of those containing actions performed to enable the performance of other actions. Consider this simple exchange (which we will see a lot more of later):

> **Q:** I want to talk to Mort. What's the combination to the Linc Lab door?
> **R:** Mort is not in the Linc Lab, he's upstairs in Ira's office.

The underlying plan involves enablement: knowing the combination to the door enables opening it, which in turn enables Q to enter the Linc Lab, which in turn enables Q ultimately to talk to Mort, assuming Mort is in the Linc Lab.

Nevertheless, Pollack's claim that actors often form invalid plans and respondents must be able to infer those invalid plans, and its corollary, that respondents must be able to evaluate inferred plans prior to choosing their response, are essential hypotheses for modeling much of cooperative behavior.

## 2.2.4 Responding to Plan-Oriented Misconceptions

Quilici *et al.* present an explanation-based approach to detecting and responding to mistaken beliefs about plans discovered during advice-seeking dialogue [49]. They focus on misconceptions about

plan *applicability conditions*, *enablements*, and *effects*. Their AQUA system is intended to produce responses that correct mistaken beliefs and identify their underlying causes.

Input to AQUA is a set of propositions representing the user's beliefs as reflected by his question. For example, given the following query:

> **Q:** I tried to remove a file with the 'rm' command. But the file was not removed and the error message was "permission denied." I checked and I own the file. What's wrong?

the corresponding beliefs would be represented as:[10]

1. `applies`(*using 'rm file', the file's removal*)
2. `enables`(*owning the file, using 'rm file', the file's removal*)
3. `causes`(*using 'rm' on the file, an error message*)

The predicates involved are defined thus:

| | |
|---|---|
| `causes(A,S)` | Executing A has S as an effect |
| `enables(S1,A,S2)` | S1 is necessary for A to have S2 as an effect |
| `applies(A,S)` | A is a correct plan for achieving state S |

It should be noted that this representation treats plans as first-class objects: the `applies` predicate is used to define the plans appropriate for achieving a given goal. The `causes` predicate is distinct from `applies` in that `causes` defines *all* effects of an action, while `applies` relates only *intended effects* to actions.

Given a set of beliefs represented in this way, AQUA follows a four-step process for each element in the set:

1. Advisor checks whether he holds the user's belief.
2. Advisor confirms a belief is mistaken by finding an explanation for why he does not hold the belief.
3. Advisor detects mistaken underlying beliefs by trying to find an explanation for why the user holds the belief.
4. Advisor presents explanations derived from steps 2 and 3 to the user.[11]

In step 2, if the advisor is unable to find a reason why he does not hold the user's belief, he may adopt that belief as his own. Learning of this sort is not addressed, however.

The paper's contribution relevant to plan evaluation concerns how the advisor searches for an explanation of the user's misconception. For each kind of mistaken belief, Quilici defines domain-independent configurations of advisor beliefs that could constitute explanations for the error. When a mistake is detected, AQUA classifies the error in its taxonomy and tests the relevant belief patterns to see if they apply. When a set of advisor beliefs is found that explains the user's error, the advisor's (correct) beliefs are presented to the user as a cooperative response. The taxonomy of explanations for misconceptions is reviewed next.

[10] The authors omit the representation of the items in italics, considering that to be extraneous detail.

[11] AQUA does not perform any NL text generation, so "presents explanations" should be understood as "produces a representation of the beliefs constituting the explanations."

### Potential explanations for not holding a belief

Because there are only three defined plan relationships, there are only three kinds of beliefs that can be unshared between user and advisor. They may disagree about whether plan is appropriate to a goal, whether a state is an action precondition, or whether an action has a particular effect.

**Unshared `applies`** If the belief `applies(A,S)` is held by the user but not by the advisor, the advisor attempts to verify that the user's belief is mistaken by first making sure that he believes (either explicitly or by deduction) `not(applies(A,S))`, and then checking if he can find another plan that *is* applicable. So, for example, if the user believes that the Unix "ls" command is used to display the contents of a file, the advisor would first make sure that he believes that "ls" is not used for that purpose, and then find out that he believes that the "cat" command is. The cooperative response would then point both these facts out.

**Unshared `enables`** If an `enables(P,A,S)` belief is unshared, the advisor verifies his belief that P is not a precondition of action A to achieve effect S, and checks for the existence of beliefs about other preconditions of A. So if the user believes that owning a file is a precondition for being able to write on it, the advisor makes sure he disagrees and then finds that he has other beliefs, such as a belief that having write permission on a file is a precondition for modifying the file.

**Unshared `causes`** Unshared `causes(A,S)` beliefs are handled similarly to unshared `enables` beliefs. The advisor verifies that he does not believe that the action A causes the state S and then finds another action that does cause S. Both advisor beliefs are presented to the user.

### Potential explanations for holding a mistaken belief

The previous set of explanations focused on how an advisor verifies discrepancies between his beliefs and those of a user. Once a discrepancy is discovered and verified, the advisor attempts to explain why the user holds the mistaken belief. The paper provides a taxonomy of potential explanations, distinguishing misconceptions about `applies`, `enables`, or `causes` beliefs.

**Incorrect `applies`** The paper lists four explanations for why a user might incorrectly believe a plan applicable to a goal. First, the plan might be used to achieve a different goal. Second, the plan might not have the desired effect.[12] Third, a necessary precondition of the plan might be unsatisfied. Fourth, the plan might *thwart* the desired goal, having an effect that denies the goal state. Each of these conditions is identified by an abstract configuration of the advisor's beliefs. The principle is that once the advisor matches the situation at hand against one of the patterns of belief, he can simply provide his (correct) beliefs about the situation as a cooperative response.

**Incorrect `enables`** If the user falsely believes that state S enables action A, the problem may be that (1) state S enables the actual enablement,[13] (2) the enablement may be unnecessary—the plan will work without the enablement in question, or (3) the enablement may be *too specific*, that is, "the user is unaware that his precondition is less general than the actual precondition for achieving his goal" [49, p. 17].[14]

[12]The distinction between a plan achieving a different goal and missing an effect seems artificial.

[13]This reveals one of the flaws in the chosen representation: type distinctions between states, actions, and plans are often blurred.

[14]This strikes me as an *ad hoc* explanation, more fallout from the awkward representation. The example given in support is the user's belief that removing a file enables creating a file. The actual precondition is sufficient space,

**Incorrect causes** Mistakes about **causes** have one of three explanations: (1) the so-called "effect from another plan,"[15] in which the action is actually used to **cause** a different effect, (2) the action has an unsatisfied precondition, or (3) "effect inferred from other effect," suggesting that one effect of the plan led the user to incorrectly believe what was in fact another effect of the plan.[16]

**Summary**

I have mixed reactions to this paper. It is hurt by the use of an awkward representation that appears to been chosen more for goodness of fit to the testbed domain than for clarity, clean semantics, and representational efficiency. As noted earlier, the authors occasionally confuse actions and states in their examples.

The paper is useful in that it suggests a simple and straightforward method for providing cooperative responses when plan-oriented misconceptions are detected: the advisor identifies his beliefs that explain why he does not share a user belief, then applies a series of tests (defined independently of any domain) that identify the advisor's beliefs explaining the user's misconception. These beliefs are then packaged up and refined into the cooperative NL response.

Unfortunately, the approach is too simpleminded for a general cooperative system. The explanation patterns substitute for general principles governing decisions about what information is relevant and what should be included in the response. There is no ability to perform other kinds of reasoning outside the plan-oriented domain. For plan evaluation, however, the taxonomy of explanations is relevant and identifies several kinds of tests that can usefully be applied to inferred plans.

### 2.2.5 Other studies

As mentioned in the introduction, several projects have focused on particular forms of cooperative behavior; many of these suggest tasks appropriate to a plan evaluator. For example, Kaplan showed how the presuppositions of a query may lead to *extensional query failure* [31]. Mays provided evidence for the slightly different case of *intensional query failure* [35]. Tests for failures such as these might well be incorporated into a plan evaluator.

While a number of the works reviewed in this section have been concerned with the generation or inference of plans, Miller's ATTENDING system (and its follow-on system HT-ATTENDING) is designed to *critique* (in NL) a physician's anesthesia management plan [40, 41]. That is, the physician presents the system with a patient's current problem and relevant medical record along with a proposed plan of action for inducing and maintaining sedation during surgery, and the system returns an analysis of the risks and benefits of that plan. In this domain, plans are represented at a level of abstraction that records only information about general operations (such as induction, intubation, and maintenance) and the particular procedures to carry out for each operation (for example, intubation can be carried out "after mask induction with cricoid pressure"). Evaluation then consists of comparing the known risks of each procedure (recorded in the system's database) against the patient's medical condition. Other parts of the system handle the structuring of the response. Pertinent to the plan evaluator is the notion that evaluation may involve analyzing the risks of the questioner's plan and marking it accordingly.

---

which may be caused by removing files. The argument is muddled by its treatment of "removing a file" as both an enablement (state) and an action. The point is that remove-file can be used to cause the enablement of create-file, but that's not the only way. This problem definitely must be addressed, but in a cleaner fashion.

[15] A great title for a plan-oriented horror flick.

[16] This is another peculiar explanation.

# Chapter 3

# Plan Evaluation

The first chapter introduced *plan evaluation* as the process of analyzing a previously inferred plan for errors. I claimed that a model of cooperative response generation that included plan evaluation could account for a significant portion of cooperative behavior. In this chapter, I will discuss plan evaluation in more detail and describe how it will be handled in the system I am developing.

## 3.1 From Inference to Evaluation

A cooperative response is one that addresses the needs and desires of the questioner. But how are those needs and desires identified? I will argue that respondents use the results of two reasoning processes—plan *inference* and plan *evaluation*—to perform that identification.

Plan inference is the process by which a respondent hypothesizes about a questioner's immediate and long-term plans and goals. A *goal* is generally thought of as some state of affairs that an agent wants to bring about. Goals can be general, like wanting to be a good cook, or they can be specific, like wanting to eat all my spaghetti without splashing sauce on my shirt. A *plan* is a sequence of actions that are performed in order to achieve a particular goal or set thereof. So I might form a plan to go to the bookstore, buy a cookbook, and memorize it, in order to achieve my goal of being a good cook, and I might plan to go to my kitchen closet, take out a bib, and put it on, or just to eat my spaghetti slowly, in order to avoid soiling my shirt.

For NLQA systems, plan inference is a problem of extracting from the questioner's utterances conclusions about his possible goals and plans. Allen showed how plan inference could be used to interpret indirect speech acts and sentence fragments [1]. For example, he demonstrated how one might reason from the utterance, "it's cold in here," spoken by a master to a servant, to the conclusion that the master wants the servant to close the window. That reasoning goes roughly as follows:

1. the servant recognizes that the statement is true and decides that it may be an indirect request;
2. the servant infers that the master does not want it to be cold in the room;
3. the servant recognizes that closing the window will make the room warmer;
4. the servant infers that the master wants him to close the window.

Sentence fragments can also be understood with the assistance of plan inference. Consider this exchange at a gas station, where Q is a customer and R the attendant:[1]

**Q:** Grant Avenue?
**R:** Six lights down. There's a BP station on the corner.

To reply appropriately to the first question, R must recognize that Q wants directions to Grant Avenue. The context of the dialogue—a gas station, where requests for directions are typical—makes this inference reasonable. It is likely that from experience the attendant has compiled a list of goals that customers are likely to have, getting directions being one of them. From knowledge of goals, the attendant could create sets of possible plan "skeletons" (plan patterns with unbound variables, cf. also Friedland and Iwasaki [17]) that customers might use to reach those goals. If a customer performs an action that fits into one of the known skeletal plans, then the goal is immediately available; the plan can then be instantiated and a reply generated. Naming a road (as in the example) might fit into one of the plan skeletons for requests for directions.

Plan inference can help a respondent determine that a questioner needs information that he did not specifically request. For example, R might believe that Q needs to know some fact P in order to carry out his plan successfully or more easily. Allen suggests that using plan inference a respondent can identify (potential) *obstacles* in a questioner's plan (cf. Chapter 2, Section 2.2.2) and then provide information that helps Q to overcome the obstacles. That analysis explains many instances of cooperative behavior, as in this example (from Allen [1]):

**Q:** When does the train to Montréal leave?
**R:** 3:15 at gate 7.

Although the departure gate is not requested, R provides it. This is done, so the argument goes, because R determines that without knowing the departure gate, Q cannot (or might not) reach his goal of boarding the Montréal train.[2]

I want to maintain a clear distinction between plan inference and the process of selecting a response. Plan inference merely yields information that helps a respondent select a suitable reply. While questioners may construct and communicate plans involving particular actions by respondents, and while respondents may infer those plans and recognize the actions that they are intended to take, the respondents are not bound to those actions. The servant could have offered to light a fire in the fireplace, or could have brought the master a blanket or sweater, if he thought that doing so would better satisfy the master. Although the servant may recognize that the master intends the window to be closed, he is nevertheless free to pursue a different course of action, as long as the same goal is achieved.[3] How plan inference fits into the larger response generation process is the subject of Chapter 4.

We see that the "plan inference hypothesis," the hypothesis that respondents use knowledge from plan inference while deciding how to answer queries, explains several forms of cooperative response behavior. But what happens when questioners have invalid plans?

What Pollack dubs the *appropriate query assumption* is the assumption that questioners accurately request information that they need to achieve their goals (cf. Chapter 2, Section 2.2.3).

[1]This is an actual exchange I had with an attendant at a Gulf station on Roosevelt Boulevard in Northeast Philadelphia. It is a very common sort of exchange in this situation.

[2]Unfortunately, the "obstacle" analysis does not satisfactorily explain other kinds of helpful additions such as the attendant's remark about the BP station. It stretches the definition of "obstacle" to say that not knowing about the BP station is an obstacle in Q's plan to find Grant Avenue. More likely R simply felt that mentioning a landmark like the BP would make it easier for Q to find Grant Avenue.

[3]It is probably the case that the servant (or any agent for that matter) is only free to pursue a different plan if it can be shown to be better in some sense than the original, e.g., more convenient, more efficient, or more expeditious.

She argues that a realistic model of cooperative interaction must abandon this assumption if useful responses are to be given to queries such as this:

> **Q:** I want to prevent Tom from reading my files. How do I set their permissions to faculty-read-only?

Problems arise if a "faculty-read-only" file permission does not exist, or if Tom is a faculty member, or if Tom is the system manager and can therefore read files regardless of access permissions. In such cases, Q's underlying plan is invalid. When Q's plan is invalid, what should a plan inference mechanism return?

Pollack's position on this question, a position with which I wholly agree, is that the result of plan inference should be a description of R's beliefs about Q's beliefs that best explains the relationships between Q's stated intentions. Thus all relevant misconceptions about the domain that Q is believed to hold should be reflected in the plans that are ascribed to Q. Therefore, a model of plan inference is needed that allows plans to contain beliefs and intentions that R recognizes to be in error. Pollack has taken the first steps in that direction.

Some instances of cooperative behavior suggest that the respondent has not only inferred the questioner's invalid plan but also has identified the source (or sources) of the error. The following example comes from transcripts of electronic mail exchanges between consultants and users of the Unix operating system at Berkeley (henceforth the "Berkeley transcripts"):

> **Q:** How come I cannot mail files from ucb-ernie and ucb-dali to ucb-cory? I was remotely logged on and it says that I was not able to send files to y:cc-48. Can you tell me how I can do this?
> **R:** The new form of mail addressing is cc-48@ucbcory. For more information, see man mailaddr.

From Q's utterances, R can infer:

1. Q wants to mail files from the machines ucb-ernie and ucb-dali to ucb-cory.
2. Q's plan includes addressing the mail to "y:cc-48."

Superficially, Q has asked to be told "how" to send the mail successfully. Such a "how" question can be answered in many ways; R could respond by describing the system's mail program, for example. Instead, R responds by informing Q that he is using the wrong addressing syntax.[4] How might this discrepancy be explained?

I claim that plan evaluation is integral to the explanation. R first infers Q's plan and then evaluates that plan to discover a disagreement in their beliefs: Q seems to believe that "y:cc-48" is a valid address syntax whereas R believes that syntax to be obsolete. The plan evaluation process serves to highlight Q's error. R may then choose to correct that error, understanding that she will thereby further Q's goals. Note that plan evaluation, like plan inference, is just another reasoning process that informs R's selection of a response, and that recognizing an error in Q's beliefs does not necessarily obligate R to address the error. If, for example, R knew that both ucb-dali and ucb-ernie had been shut down and sold, she need not mention in her response that Q was using the wrong addressing syntax.

For another (realistic but constructed) example, consider this dialogue:[5]

[4] R needs special domain knowledge in order to recognize that "y:cc-48" is an obsolete address syntax in which "y" designates a machine and "cc-48" a user.

[5] The processing of several variations of this example will be explored in detail in Chapter 4, Section 4.4.

**Q:** I want to talk to Mort. What is the combination to the Linc Lab door?
**R:** The door is not locked.

Plan evaluation again seems to account for R's (perfectly reasonable) response. R first infers Q's plan, the relevant portion of which is:

1. because Q wants to know the combination to the Linc Lab door, he probably wants to unlock that door;

2. because Q wants to unlock the Linc Lab door, he must believe that it is locked.

Plan evaluation highlights the discrepancy in belief—R does not believe that the Linc Lab door is locked. R may then decide to point that out to Q, allowing Q to continue executing his plan to enter the Linc Lab and see Mort.

Plan evaluation becomes an important reasoning step when questioners are allowed to form incorrect plans and make inappropriate queries as a result. The next section discusses how plan inference and evaluation will be handled in this thesis.

## 3.2 Plan Evaluation In This Thesis

Designing a plan evaluation system "right," by developing and implementing a formal theory, is a task that would require at least a full thesis' worth of effort. I will not undertake that project here because my primary interest is in the architecture of CNLQA systems. I will aim instead to show that the information we could reasonably expect a proper plan evaluation mechanism to produce can be useful in selecting a cooperative response. In order to show *that* plan evaluation data is useful, I will propose a system architecture that shows *how* that data could be used during cooperative response generation.

Yet this work must be based on *some* computational model of plan evaluation if I am to ensure that the plan evaluation data I use can reasonably be produced mechanically. I cannot just hand-code some quantity of information and claim that it could have been produced by a plan evaluation system.

Therefore, I plan to develop the thesis using a limited representation for plans, limited in that only a subset of the kinds of plans that real-world agents adopt will be encodable. Several simple kinds of errors will be representable, and I will design and implement a simple mechanism for inferring plans and annotating them with marks that describe the errors. The only claim I will make about the mechanism is that it can infer, in some general way, a variety of plans with and without errors and annotate them accordingly.

If I can show *that* a simple plan evaluation model provides useful information for cooperative response generation and *how* that simple model can be incorporated into a CRG system, then the conclusions will hold for more expressive representations and more powerful plan evaluation mechanisms. If crude plan evaluation knowledge is useful in CRG, then more refined knowledge should be at least as useful, and probably more so.[6]

The next subsections will sketch out the plan evaluation model I will develop for use in this thesis. First, I will describe the language I will be using to represent plans. Next, I will discuss some concerns about plan inference and sketch a couple of possible approaches to the design of an inference mechanism. Finally, the design of a plan evaluation system will be outlined.

[6]Of course, the question of how much plan evaluation data is *enough* under different circumstances is not considered.

### 3.2.1 A simple representation for plans

I base my representation for plans on Allen's temporal calculus [2]. I will briefly review the formalism here; for details the reader is referred to the journal article.

#### Allen's temporal calculus

Allen's calculus is a typed first-order logic that incorporates time intervals as objects in the language. *Properties* are logical propositions that hold over intervals of time. The basic predicate is HOLDS, where HOLDS$(p,t)$ is true iff the property $p$ is true over the interval $t$. Thirteen mutually-exclusive relationships are defined between time intervals, as shown in Figure 3.1.

| Relation | Symbol | Inverse | Pictorial |
|---|---|---|---|
| X before Y | < | > | XXX YYY |
| X equal Y | = | = | XXX<br>YYY |
| X meets Y | m | mi | XXXYYY |
| X overlaps Y | o | oi | XXXX<br>YYYY |
| X during Y | d | di | XXX<br>YYYYYY |
| X starts Y | s | si | XXX<br>YYYYYY |
| X finishes Y | f | fi | XXX<br>YYYYYY |

Figure 3.1: The thirteen temporal relationships

The IN predicate expresses the relationship in which one interval is entirely contained in another interval:

$$\text{IN}(t_1,t_2) \Leftrightarrow \text{DURING}(t_1,t_2) \vee \text{STARTS}(t_1,t_2) \vee \text{FINISHES}(t_1,t_2).$$

Using IN, a critical characteristic of the HOLDS predicate can be defined, namely, that if a property $p$ holds over an interval $T$, then $p$ also holds over all subintervals of $T$, as follows:

$$\text{HOLDS}(p,T) \Leftrightarrow (\forall t\ \text{IN}(t,T) \Rightarrow \text{HOLDS}(p,t)).^{7} \quad \text{(H.1)}$$

The calculus defines two kinds of *occurrences*: *events* and *processes*. The OCCUR predicate is used to represent instances of events. OCCUR$(e,t)$ is true iff event $e$ occurred over the interval $t$. Events in this logic cannot be decomposed:

$$\text{OCCUR}(e,t) \wedge \text{IN}(t',t) \Rightarrow \neg\text{OCCUR}(e,t').$$

This says that events occur over the smallest intervals possible—to say that an event occurred over some interval $t$ is to say that the event did not occur over any subinterval of $t$. For example, unlocking a door over some interval $t$ cannot be completed over only some part of $t$, it requires all of $t$.

[7] Allen also defines a stronger version of this axiom, used in later development:
HOLDS$(p,T) \Leftrightarrow \forall t\ \text{IN}(t,T) \Rightarrow (\exists s\ \text{IN}(s,t) \wedge \text{HOLDS}(p,s))$. (H.2)
Allen provides a proof that (H.1) can be derived from (H.2) plus the definition of IN.

The OCCURRING predicate is used to represent processes. OCCURRING($p,t$) is true iff process $p$ was occurring over interval $t$. Unlike events, processes need not be occurring over all their subintervals:

OCCURRING($p,t$) $\Rightarrow \exists t'$ IN($t',t$) $\wedge$ OCCURRING($p,t'$).

That is, a process $p$ must be OCCURRING over at least one subinterval $t'$ of $t$. Thus, I may be engaged in the process of walking over some time interval and still be allowed to pause during that time for a rest.

The ECAUSE predicate represents causality of events. ECAUSE($e_1,t_1,e_2,t_2$) is true iff the occurrence of event $e_1$ over interval $t_1$ caused the event $e_2$ to occur over $t_2$. ECAUSE is a transitive, anti-symmetric, anti-reflexive relation.

Actions are events or processes that are initiated or performed by animate agents. ACAUSE($a,o$) represents an occurrence caused by an agent. The action in which John closes a particular door would be written as

ACAUSE(John,CLOSE-DOOR(DOOR359)),

where CLOSE-DOOR(DOOR359) is an event in which DOOR359 becomes closed. A particular instance of John closing DOOR359 would be represented using OCCUR:

OCCUR(ACAUSE(John,CLOSE-DOOR(DOOR359)),$t$)

The *generates* relationship, originally defined by Goldman [21] and later used by Pollack [47], is also at home in this logic. As defined by Allen, the predicate GENERATES($a_1,a_2,t$) is true iff action $a_1$ generates $a_2$ over interval $t$. But this definition does not allow generic generation relationships to be defined. I suggest instead that GENERATES be a two-place predicate defining a generation relationship between two occurrences. GENERATES as an argument to OCCUR may then define an instance of one action generating another.

### Preconditions and effects

Allen's calculus provides a language for defining properties, events, processes, and actions. For example, we can state the necessary and sufficient conditions for a MOVE-TO event:

Necessary and sufficient conditions for
OCCUR(MOVE-TO(agent,destination),T):
$\exists t_1,t_2,s$
MEETS($t_1$,T) $\wedge$ MEETS(T,$t_2$) $\wedge$
HOLDS(at(agent,$s$),$t_1$) $\wedge$
HOLDS(at(agent,destination),$t_2$) $\wedge$
$s \neq$ destination

Now we can use MOVE-TO to represent an agent moving to a specific place:

OCCUR(MOVE-TO(John,Linc-Lab),$t$)

represents John going to the Linc Lab. To represent that John is moved to the Linc Lab "by himself" (that is, he's moving under his own power), we use ACAUSE:

OCCUR(ACAUSE(John,MOVE-TO(John,Linc-Lab)),$t$)

When constructing and reasoning about plans, it is convenient to have an explicit representation for the *preconditions* and *effects* of actions. In terms of the temporal logic, a precondition is a property that must hold prior to an action in order for that action to succeed. An effect is a property that will hold immediately after an action, if that action is successfully executed.

In the planning literature, a distinction is often made between preconditions and *applicability conditions*. Applicability conditions have been described either as predicates that must hold in order for an action to be well formed—rocks cannot eat—or as preconditions that cannot be affected by the planner—a train must be in the station before it can be boarded. The first use, applicability conditions as formedness constraints, is unnecessary in a typed calculus. The second use allows some efficiency in the inference mechanism, as it need not explore the territory below applicability condition nodes in the plan graph. However, giving a precondition the special status of "applicability condition" must be done with care, because one agent's applicability condition is another's precondition. A passenger cannot affect a train's schedule, but the conductor can. Some cooperative behavior definitely turns on a respondent's ability to affect the world in ways that the questioner cannot. Henceforth, I will use "precondition" to refer to both true preconditions and applicability conditions, only using the latter term when the distinction is crucial.

Although preconditions and effects can be derived from the necessary and sufficient conditions that define actions, it is useful to have separate predicates to represent and reason about them explicitly. I define PREC as follows:

$$\begin{aligned}&\text{PREC}(p,e) \Leftrightarrow \\ &\quad (\forall t\ \text{OCCURS}(e,t) \Rightarrow \\ &\qquad \exists t_1\ \text{HOLDS}(p,t_1) \wedge \\ &\qquad\quad (\text{MEETS}(t_1,t) \vee \text{OVERLAPS}(t_1,t) \vee \text{DURING}(t,t_1)).\end{aligned}$$

That is, whenever event $e$ occurs over interval $t$, property $p$ either must hold over some interval that MEETS $t$ or OVERLAPS $t$, or $e$ must occur DURING the interval that $p$ holds. EFF is defined similarly:

$$\begin{aligned}&\text{EFF}(p,e) \Leftrightarrow \\ &\quad (\forall t\ \text{OCCURS}(e,t) \Rightarrow \\ &\qquad \exists\ t_1 t_2\ \text{HOLDS}(\neg p,t_1) \wedge \text{HOLDS}(p,t_2) \wedge \\ &\qquad\quad (\text{MEETS}(t_1,t) \vee \text{OVERLAPS}(t_1,t)) \wedge \\ &\qquad\quad (\text{OVERLAPS}(t,t_2) \vee \text{MEETS}(t,t_2)).\end{aligned}$$

This states that $p$ is an effect of $e$ iff whenever $e$ occurs, $p$ is false during some immediately prior or overlapping interval and true during some immediately following or overlapping interval.

This concludes the description of the plan representation language.

### Computability

Vilain and Kautz have shown recently that both determining consistency and consequences of statements in the interval algebra may be NP-hard [54]. The trouble arises when the relationship between two intervals cannot be completely defined by one of the thirteen simple relations (cf. Figure 3.1). Consider the relation vector [< m o], the disjunction of the three relations BEFORE, MEETS, and OVERLAPS. If $t_1$ [< m o] $t_2$, then the starting point of interval $t_1$ strictly precedes that of $t_2$, and its ending point strictly precedes that of $t_2$. However, there is ambiguity as to the relation between $t_1$'s ending point and $t_2$'s starting point.

When such non-primitive temporal relations are needed to properly characterize situations, the reasoning procedures become computationally intractable. Vilain and Kautz go on to show that a point-based temporal algebra is computationally tractable, and that a fragment of the interval algebra (including some non-primitive relations) can be directly encoded in the point language and thus brought back within the realm of the tractable. As far as my own work is concerned, I will limit my studies to situations that can be adequately represented using only the relations in the tractable portion of interval calculus.

### Notation

Throughout the rest of this paper, I will use a more perspicuous notation for properties and actions. For properties, I will omit the HOLDS predicate and simply append a temporal indicator, e.g.,

ON(block3,table49)< $t$ >

is the same as

HOLDS(ON(block3,table49),$t$).

Actions by agents will be notated similarly, with

PUT-ON(Fred,block3,table49)< $t$ >

meaning the same thing as

OCCUR(ACAUSE(Fred,PUT-ON-EVENT(block3,table49)),$t$).

That is, for every action like PUT-ON, there is a corresponding event or process which can be the argument to ACAUSE. This accords with Allen's conjecture:

> It is hypothesized that every action can be characterized as an agent ACAUSEing an occurrence. [2, p. 139]

I will only resort to the full notation when the shortened notation is ambiguous.

### Summary

Because many of the more interesting kinds of cooperative responses address plan failures involving time, it makes sense to use a representation for plans in which time relationships are explicit. As long as temporal relations are restricted to the tractable subset, a polynomial time constraint-propagation algorithm exists to compute the consequences of a set of temporal assertions [54].

Of course, by adding temporal knowledge into the plan representation, complexity is increased. Without a notion of time, preconditions could be idealized as propositions that were either always true or always false. With a notion of time, the executability of actions must be evaluated more carefully; now an evaluator must ensure that preconditions hold *at the time of the action.* But the latter sort of reasoning is much more realistic—it is worth the effort to try to accommodate it in the design of real systems.

## 3.2.2 Inference procedure

Because plan inference is of interest only to the extent that it relates to plan evaluation, I will not attempt to develop a particularly sophisticated or innovative inference mechanism. I will mostly follow the inference procedure outlined by Allen [1]; however, I will have to make some modifications to incorporate temporal reasoning and plan evaluation and to rule out implausible ill-formed plans. This section discusses some related concerns.

### Coping with the search space

Plan inference is a search problem, the search space being the set of all possible plans in the domain. The heuristics used in Allen's system restrict the search space to the set of all possible *valid* domain plans, as exemplified by Heuristic H1:

**H1** Decrease the rating of a partial plan if it contains an action whose preconditions are false at the time the action starts executing.

The application of this heuristic tends to remove from further consideration any plan containing actions with unsatisfied preconditions.

The *raison d'etre* of plan evaluation is the fact that people often unwittingly devise plans that may fail to achieve their goals. But once agents are allowed to hold invalid plans, heuristics such as H1 are no longer useful, and the search space—the set of all possible plans—becomes too huge to cope with. Yet it seems that for any given goal in a particular situation there is only a small number of invalid plans which one person might reasonably expect another to have. But on what basis can we discriminate the invalid but reasonable plans from the invalid and unreasonable ones?

The solution, I think, will require that inference rules be tested for plausibility as they are applied. Specifically, after matching the antecedent of a plan inference rule to a partial plan but before expanding the plan according to the consequent of the rule, one or more tests might be performed to judge the reasonableness of the consequent in the given context.[8] For example, consider Allen's *precondition-action rule*: if the system believes that Q wants proposition P, and P is a precondition of action $\alpha$, then infer that Q intends to do $\alpha$. When the plan inference mechanism has matched the *precondition-action rule* against a particular proposition P and action $\alpha$, the plausibility that Q actually intends to do $\alpha$ is tested. That may require answering questions like: does R believe that Q knows *about* $\alpha$? Does R believe Q knows that P is a precondition of $\alpha$? Does R believe that Q knows the effects of $\alpha$? In contrast with Allen's heuristic H1, does R believe that Q believes that all of the preconditions of $\alpha$ hold?

As I have just presented them, the reasonableness tests depend on the availability of a store of *a priori* knowledge about the user—a so-called *user model.* In theory a user model would be ideal in helping a plan inference mechanism eliminate unlikely plans from consideration. In practice, however, their usefulness is uncertain. For one thing, it is far from clear how much knowledge about a user can be reasonably expected to appear in a user model. A related problem is acquisition of that knowledge: how does the knowledge get into the user model?

Kass has identified two techniques for user model knowledge acquisition: *explicit* and *implicit* [32]. Explicit knowledge acquisition is the most familiar: system developers manually encode the knowledge in the user model. Using implicit knowledge acquisition, however, facts about the user are deduced during interaction and the user model is updated dynamically. Kass has proposed a set of plausible inference rules for implicit knowledge acquisition.

The thrust of Kass' thesis seems to me to be right-minded. Intuition agrees that both interaction style and content provide definite clues about an interlocutor's knowledge. However, the thesis raises several difficult questions which are left unanswered, the most important of which is: how can we be sure that implicit acquisition techniques will have yielded the necessary information by the appropriate time? It's one thing to say that we can use implicit acquisition techniques to deduce facts about a user's knowledge; it's quite a different matter to claim that doing so is *useful*, that solely on the basis of those techniques we can get all the information we need when we need it, say, for a plan inference mechanism in a cooperative system.

Another possible way to constrain the search space is simply to allow certain preconditions to be left unsatisfied or certain constraints unmet during the inference process. Consider, for example, this exchange:

> **Q:** I want to talk to Kathy. What's the phone number at the hospital?
> **R:** Kathy's not at the hospital, she's at home. Her number there is 123-4567.

Suppose our domain knowledge tells us that:

1. knowing the phone number of a location is a precondition to telephoning that location;

[8] Of course, an alternative is to make the plan inference rules "smarter," building the reasonableness tests into their antecedents.

2. telephoning a location enables one to talk to people at that location.

R might first reason backward from knowledge of Q's goal—to talk to Kathy—to infer that Q might want to telephone Kathy's location. Call this partial plan 1. Next, R could reason forward from knowledge of Q's request, inferring that Q wants to telephone the hospital. Call this partial plan 2. Joining partial plans 1 and 2 would involve binding the variable for "Kathy's location" in plan 1 to "the hospital" from plan 2. From R's point of view, that is an incorrect binding—Kathy's location is "at home." But if R just assumes that Q believes Kathy is at the hospital, she can conclude the inference process. Type checking may be able to prevent ludicrous bindings, e.g., telephone calls can only be made to locations, and the hospital is a location, so the binding, although incorrect (to R) is at least plausible.

It might be possible to get quite a bit of mileage out of this method, avoiding user models altogether. Of course, this still assumes that Q and R share all their knowledge of domain actions, a dubious assumption at best. But once we allow discrepancies in basic domain knowledge, a user model seems inescapable. At the moment, I think it best to make the simplifying assumption and leave the trials and tribulations of user modeling aside.

Next, I will make a few comments on the inference mechanism I intend to design and implement.

#### Task of inference mechanism

The inference mechanism will attempt to find a plan that reflects the questioner's beliefs that underlie his utterances. In constructing the plan graph, territory outside the inference chains will not be explored. For example, application of a rule like Allen's *precondition-action* rule adds an action node to the graph upon finding a node that is one of the action's preconditions. After applying such a rule, no attempt will be made to examine the structure below the action's other preconditions, if any. Nodes will be created corresponding to those preconditions (so that they may be tested by the evaluator), but no resources will be expended trying to figure out how a precondition might be satisfied except in an attempt to find a connection to another partial plan.

The chosen representation language allows actions in a plan to be related either by enablement or by generation. As long as preconditions, effects, and generation relationships are represented explicitly in the domain model, the inference mechanism should be able to construct plans containing actions related in either way.

As the representation language is a temporal calculus, action definitions must include temporal constraints. The inference procedure must check these constraints as partial plans are being built.

### 3.2.3 Detectable errors

There are several different kinds of errors that a plan evaluation component should be able to detect. These errors fall into one of two groups: those that can be detected without using a model of the user's knowledge, and those for which knowledge of the user is essential to detection.

#### Unexecutable actions

An action is judged *unexecutable* when the system believes it cannot be successfully executed at its intended time. An instance of a BOARD-TRAIN action, intended over interval $t$, is considered unexecutable if, for example, the train to be boarded is not in the station over interval $t$. The evaluator should mark an action node unexecutable if one or more of its preconditions will not be satisfied over the appropriate interval. If a failed precondition happens to be an applicability condition, that fact should be noted since cooperative behavior is likely to differ if an applicability

condition (which cannot be affected by the user) fails rather than a plain precondition. Knowledge of the user is not required to detected unexecutable actions.

### Unnecessary actions

It is not uncommon for people to attempt actions whose effects already hold. This usually happens when the results of an action cannot be easily perceived. The perfect example is a locked door: just from looking at a door, one may not be able to determine whether it is locked or not. So one might attempt to unlock a door that is already unlocked. The plan evaluator should mark an action node unnecessary if its effects will hold after the intended action interval regardless of whether the action is performed. User model knowledge is not necessary to detect these kinds of errors.

### Type failures

Type failures arise when a user attempts to perform an ill-constructed action. The paradigm case is Pollack's example of a user wanting to set a file's access code to "faculty read-only." If such a permission does not exist, and the plan inference mechanism is able to construct plans flawed in this way, then a type failure will exist: the SET-PERMISSION action expects an argument of type FILE-PERMISSION and "faculty read-only" is not of that type. Some type failures may indicate object-related misconceptions (cf. McCoy [37]). Type failures are detectable when a typed representation language is used; no user model knowledge is necessary.

### Constraint failures

Constraint failures include any non-type-related errors introduced during the inference process. Constraint failures are typically generated when the plan inference mechanism posits, in an attempt to link two partial plans, a belief of the user that the system does not believe to be true. Constraint failures are only possible if allowed by the inference mechanism.

The next few errors only seem detectable if an appropriately knowledgeable user model is available for use by the evaluator.

### Unknown preconditions

It happens that from time to time, people don't know all the preconditions of an action. This is not a problem when an unknown precondition will nevertheless hold prior to an action. If, however, a precondition will not be satisfied and the user does not know of it, the cooperative response may be affected. For example, suppose you tell me that you intend to set the access permissions on a computer file to "private" in order to prevent Ira from reading it. Suppose further that I know that by dint of being the system administrator, Ira is unaffected by file permissions. If I believe that you know Ira is the system administrator, then I might guess that you don't know that system administrators are able to bypass normal file protection mechanisms. My response might be to inform you of that fact. If, however, I believe that you don't know Ira is the system administrator, then I might respond by telling you that and assume you know the rest of the story. In both cases, the problem is a failed precondition. The responses differ based on whether the user knows about the precondition.[9]

[9] Pollack's SPIRIT system could perform this kind of reasoning. When a precondition didn't hold, SPIRIT checked a user model and gave different responses depending upon whether or not the user knew about the precondition.

**Invalid preconditions/effects**

Users will occasionally have beliefs about domain actions that are quite different from the system's beliefs. They may incorrectly believe that a particular state is a precondition or an effect of an action. For example, a user might believe that special tokens are required for bus fare, when in fact the buses accept normal coins and bills. Or a user might believe that sending electronic mail to user "bboard" has the effect of posting an article on the computer's general bulletin board, when in fact the "postnews" program must be used. In order to be able to attribute such erroneous plans to a user, a plan inference mechanism seems to require access to knowledge about the user's false beliefs. Knowing that a user has a false belief about a domain action's preconditions or effects may cause a respondent to attempt to correct the user's relevant knowledge.

**Unknown effects**

Domain actions often have side effects of which a user may be unaware. This is especially true in computer operations. For example, the Unix "mv" command can be used to change the name of a file. If, however, one uses "mv" to rename "baz" as "bletch," and "bletch" already exists, then the contents of "bletch" will be silently deleted and replaced with those of "baz." If the respondent is able to identify the effects of an action that are unknown to the user, she could provide warnings when appropriate.

### 3.2.4 Design of the evaluator

A plan evaluator would be designed most efficiently, I think, if its operation was interleaved with the inference process. Specifically, the evaluator should be called after each plan inference rule is applied and allowed to inspect the affected partial plans. The evaluator can then perform the appropriate tests and make any necessary annotations to the partial plan graphs. The inference mechanism, once resumed, must ensure that any changes made to the graph structure (as a result of linking partial plans) do not cause the evaluator's comments to be lost.

For a specific example, consider an intermediate stage in the processing of this familiar query:

**Q:** I want to talk to Kathy. What's the phone number at the hospital?

Suppose that the inference mechanism has completed constructing partial plans 1 and 2, structured as before (cf. Section 3.2.2). Finally, assume that TALK-TO is a domain action, defined so that telephoning the location of the person to TALK-TO enables (is a precondition of) the execution of the TALK-TO action. Matching the antecedent of the *precondition-action* rule in this context binds the precondition P to the node for "telephone hospital" and the action $\alpha$ to the node for "talk to Kathy." At this point, the evaluator could be called, given P and $\alpha$, and told PREC(P,$\alpha$).

The evaluator is now in the position to perform any of the relevant tests. It can check whether any preconditions of $\alpha$ will not hold over the proper interval. It can check constraints and discover that the knowledge base shows that Kathy is not at the hospital. In this case, the evaluator can annotate the node for $\alpha$ indicating that the constraint has failed. Note, by the way, the complexity of the constraint: for action

TALK-TO(Q,Kathy)

to succeed, the precondition

CALL-BY-TELEPHONE(Q,LOCATION(Kathy))

must be satisfied. However, the plan graph suggests that Q actually intends to perform

CALL-BY-TELEPHONE(Q,hospital).

Looking up LOCATION(Kathy) does not yield the result "hospital," hence the failure. The constraint is on valid instantiations of the precondition to TALK-TO.

### Propagation effects

The plan evaluator is an advisor; its job is to bring the questioner's errors to the attention of the response processor. It must not overwhelm the response processor with lots of details, it should identify only the root cause of each plan failure. In particular, error propagation effects must be avoided.

The problem with errors in plans is that they tend to cause other errors. Suppose Q plans to attend a meeting with Mort in the Linc Lab. He might plan to GO-TO the Linc Lab, OPEN the door, ENTER the lab, and MEET with Mort. The effect of OPENing the door is that the door is in the OPEN state, which is a precondition to being able to ENTER the lab.

Now suppose the door is locked. The evaluator should notice the OPEN action and discover that one of its preconditions—an unlocked door—will not hold. That causes the OPEN action to be unexecutable. Since unexecutable actions cannot achieve their effects, the door will not be open prior to the ENTER action and thus it, too, is unexecutable. Because Q cannot ENTER the Linc Lab, the goal will not be reached. The error—the unanticipated locked door—generates, domino-like, other errors, and seems to invalidate the entire plan.

This is undesirable behavior. If the evaluator acted like this, the response processor would have to sort through all the errors in order to select the ones to advise the user about. A better approach is to identify only the "first dominoes," the errors from which all other errors are derived. The response processor can then decide on its own if the error propagation details are relevant.

### By-products of processing

It is possible that potentially useful information will be generated as a by-product of the plan evaluator's operation. If so, that information should be somehow made easily accessible to the response processor.

To illustrate, consider again the processing just before joining the two partial plans involving "call hospital" and "talk to Kathy." In order to test whether the LOCATION(Kathy) constraint is satisfied, the evaluator may need to request from the knowledge base the current believed value for LOCATION(Kathy). That result is "at home." The evaluator determines that "at home" is not the same as "the hospital," and marks the node to highlight the constraint failure. But as part of its work, the plan evaluator had to call upon (potentially expensive) resources to find Kathy's location. Rather than wasting those resources, Kathy's believed location could be attached to one of the nodes (presumably the one containing the mark indicating constraint failure) in some way that could be understood and used by later processes.

## 3.3 Summary

I argued in this chapter that the data yielded by plan inference and evaluation is important, both in understanding a questioner's utterances and in selecting a response that is both relevant and useful. Plan inference helps a respondent to identify the questioner's goals and thus enables her to produce responses that address those goals. Plan evaluation highlights for the respondent the flaws that may cause a questioner's plan to fail. When questioners' plans are believed to contain errors, cooperative respondents adjust their replies accordingly, providing either more information or different information as needed to help the questioners achieve their goals.

For plan evaluation to be meaningful, the underlying inference system must be able to produce plans that contain typical kinds of errors. Following Pollack, plans are viewed as configurations of the respondent's beliefs about the questioner's domain beliefs and intentions. Plan inference components in cooperative NLQA systems therefore should produce structures, representing the questioner's beliefs and intended actions, that explain the observed utterances.

Designing an adequate plan inference mechanism, one that can introduce motivated errors into plans, is difficult as the search space is quite large. Two possible approaches have been suggested: (1) use a user model, and (2) allow some constraints to fail during the inference process. The former, in theory the most accurate and reliable way to reduce the amount of search necessary, presents several discouraging practical difficulties. The latter approach is less efficient and more prone to err, but in many cases may produce equally good results without requiring vast amounts of knowledge. This approach deserves more careful consideration.

I discussed several general categories of errors that a plan evaluation component should be able to detect. Assuming a capable underlying inference mechanism, the more common errors can be detected by reasoning on the system's knowledge alone. More esoteric errors seem to require a store of *a priori* knowledge about the user.

Although there are only a few basic kinds of error, the range of appropriate response behavior is not similarly constrained. There may be many different reasons why a questioner and a respondent disagree about a domain fact. Each reason is likely to affect the choice of a response. Though there are probably general strategies appropriate in each situation, specific response options will vary between domains.

I will use Allen's interval-based temporal calculus to represent domain knowledge about states (properties), events, actions, and processes. I defined the predicates PREC and EFF so that preconditions and effects of actions can be represented explicitly. Applicability conditions are preconditions that are beyond a planning agent's control; these will have to be represented explicitly.

I suggested that the plan evaluator's operation should be interleaved with that of the plan inference mechanism. The problem of error propagation was discussed; in order to best advise the response processor's choice of cooperative response, the evaluator should make sure that only root causes of plan failures are identified.

Because plan evaluation is of interest only to the extent that it supports cooperative response generation, this research makes no attempt to develop a fully adequate theory of plan inference and evaluation. Instead, a limited computational model will be used, allowing plans containing common kinds of errors to be produced mechanically.

In order to show *that* plan evaluation data is useful in producing cooperative response behavior, I will show *how* it may be used in that endeavor. The next chapter presents an architecture for cooperative NLQA systems that incorporates a plan inference and evaluation component.

# Chapter 4

# The Architecture of a Cooperative Respondent

The preceding chapters have set the stage for discussion of the final topic in this proposal, the architecture of a cooperative response generation system. In this, the penultimate chapter, I will outline and motivate an architecture design upon which I propose to base a prototype CNLQA system implementation (cf. Chapter 5).

In the next section, I discuss two characteristics of cooperative response behavior that motivate my architecture design. I will then describe the proposed architecture in detail and present an example intended to show how a system so framed could produce a cooperative response.

## 4.1 Characteristics of Cooperative Response Generation

What characteristics of the CRG process might be used to motivate the design of a CRG system? Analysis of various transcripts of natural language question-answer dialogues suggests two: (1) CRG systems should be able to explicitly reason about and choose among the different response options available to them in a given situation, and (2) CRG systems should be able to *reflect* on their selections of response content before producing any output. Some choices of response content motivate others; the process of reflection allows respondents to detect a need to explain, justify, or clarify information they have already decided to convey.

Before continuing, let me define some terms. A *response option* describes a specific communicative act that might be performed as part of a response. Given a question like "is P true," one response option might be "inform the questioner of P's truth value." A respondent might choose to *exercise* a response option, meaning that she decides to perform the act described by the response option in her response. Exercising the option "inform the questioner of P's truth value" might lead to the appearance of "P is true" in a natural language response, assuming P were in fact true. Thus, response options are goals that a respondent wants to achieve by her response. The actual response is produced by forming and executing a *response plan* that achieves the goals specified by the response options.

The main arguments in this section will make use of two examples shown in Figure 4.1, extracted from the Berkeley transcripts.

## Example 1

**Q:** Do you know if there is any network link to ucb-euler? I want to send some mail over there.

**R:** Gary, just do this: mail ruby!euler!(login name). Let us know if it doesn't work. Euler is only reached thru the ruby machine.

## Example 2

**Q:** Is there a way to send mail to ucb-euler from ucb-cory?

**R:** Yes, it is letter y on the Berknet. So mail user@y.CC. If you have further problems with it, mail to serge@cory. He is the euler system manager.

Figure 4.1: Two Cooperative Exchanges

### 4.1.1 Selecting among multiple response options

To begin with, the utterances comprising a query appear to license various response options, only some of which are exercised in any actual response. In Example 1, Q has requested to be informed *if* a network link exists between his computer and ucb-euler. There are several pieces of information which R might consider conveying in her response to this question. She could certainly give the *direct answer*, telling Q that there is in fact a network link between the two machines. In an effort to be helpful, she might tell Q *how many* such links there are, assuming a count of this kind is meaningful. Or she might tell Q *what kind* of links there are, e.g., a high-speed ethernet link, a low-speed phonenet connection, and so forth. Recast as response options, R might identify her options as "inform Q that a network link exists," "inform Q of the count of network links," and "inform Q of the type of each network link."

R's possible response options follow from general principles of interaction and reasoning, based on beliefs about Q's goals and intended actions. On general principle, for example, queries that ask "whether P" can always be answered by informing Q of P's truth value. In response to "is there a P," one might include a count of the Ps if there is more than one—"over-answering" the question, in the sense of Wahlster [55]. Reasoning on her beliefs about Q's plan, R may be able to identify potential obstacles that can be eliminated by providing Q with some information.

In Example 1, R does not produce all possible response options. In fact, she does not even explicitly give the direct answer. The point is that the direct answer can be deduced from the response: there clearly is some network connection between the machines, at least one that permits the transmittal of mail messages. So either R has decided to convey the direct answer implicitly, or she has decided to leave it out, with the direct answer being implied purely by happenstance.

I take this as evidence that R considered her plausible response options and decided which to include in the final response. In terms of a computational model, R first identifies the available response options and then decides which ones to actually exercise.

There are different bases for rejecting (deciding not to exercise) a response option. For example, exercising one response option may make another unnecessary—replying "there are three Ps" when asked "is there a P" makes the direct answer of "yes" unnecessary. Alternatively, one response option may, in the given circumstance, be considered more important than another: correcting a misconception evident from Q's query may be more important than answering his question. Finally,

a respondent may have to reject response options simply to avoid a lengthy reply.

The Berkeley transcripts contain many examples of open-ended queries in which Q primarily describes his goal, leaving it up to R to decide how to respond:

> **Q:** I'm using a hardcopy printing terminal on a dialup line. In ex[1] I use the space bar to indent lines in a program I'm entering. After I indent on one line, the next line automatically indents the same number of spaces, but I don't want it to. I got out of ex and then went back in and the indent was still set where it had been. Logging out removed the automatic indenting, but that's a hard way to work! Any suggestions?

Q's query ("any suggestions?") is a general request for help with the described problem. Here is another situation in which, in choosing her response, R may have to weigh different options. For example, she might recommend that Q switch to a different editor, or if automatic indentation indicates an improperly configured terminal, R might want to point that out.

Based on these arguments, I propose that a rating/selection phase be part of a computational model of cooperative response generation. The system should first identify all the response options that it deems potentially relevant and useful. These options are then rated according to the kinds of criteria mentioned earlier. The ratings then influence which options are chosen to be exercised in the system's cooperative response.

I want to be clear that I believe rating/selection is a necessary attribute of CRG *systems*, not necessarily part of a model of human cooperative activity. Given a question, NLQA systems are not constrained as much as people are to begin responding immediately. They should be more circumspect in their choice of response options, and take more care to be brief yet informative, not to mislead, and to appropriately justify their assertions or directives. The rating/selection process enables CRG systems to be as cooperative as they can and need to be.

### 4.1.2 Reflecting on earlier decisions

The argument for including reflection in a computational model of CRG is based on Example 2 (Figure 4.1). Notice that R asserts:

> He [serge@cory] is the euler system manager.

What might have motivated R to make this statement? There doesn't seem to be anything in Q's request, explicit or implicit, that suggests a need to be told anything about serge@cory or about euler's system manager. To account for this phenomenon, first consider the immediately preceding statement:

> If you have further problems with it, mail to serge@cory.

Why might R have included this statement in her response? Since the goal here is the development of a computational model of R's response, a better question is: what process could explain why R made the above statement part of her response?

My analysis goes as follows: first, we can assume that R inferred Q's goal of sending mail to ucb-euler, because the first part of her response helps Q reach that goal by informing him of the correct electronic mail address syntax. That is, she chose to exercise the response option "inform Q how to send mail to ucb-euler." R's next statement, "if you have further problems...," is a further attempt to help Q send mail successfully to euler. Several explanations for its appearance exist, including:

[1] In Unix, "ex" refers to a line-oriented text editor.

1. R believes that Q may still have problems (not involving incorrect address syntax) sending mail to euler, and therefore needs to know of a better way (better than asking R) to deal with those problems.

2. R is unsure that user@y.CC is in fact the correct address. However, she knows that serge@cory definitely knows the right way to send mail, so she points Q to him.

3. As a matter of policy, R routinely directs users to those people with the most expertise in the given problem area.

It seems that R's second utterance is still part of her effort to help Q reach his goal. She recognizes his goal, identifies what she takes to be his mistake (he was using the wrong address syntax), corrects his mistake, and then, allowing that other difficulties may arise, she points him to a better source of information. The presence of her third (last) utterance, though, seems to have a different explanation.

Looking at its effect on Q, "serge@cory is the euler system manager" *explains* R's second statement which mentions serge@cory for the first time. Apparently, having decided to refer to serge@cory, R realizes that she should also explain who he is.

This process I call *reflection*, to capture the idea that after selecting an initial set of response options to exercise, a respondent "reviews" or "reflects" on those choices and may be able to identify new response options that are suddenly relevant, relevant by dint of the information to be conveyed by response options already chosen. So some response options are chosen because they address the questioner's goals, plans and needs, while other response options are selected to justify, clarify, or explain options that the respondent has already decided to exercise.

Through reflection, a respondent also may decide to generate a new plan for the questioner and communicate the important details to him. Consider this example:

**Q:** The Linc Lab laserwriter is down again. What's Ira's office phone number?
**R:** Ira's not in his office. Call Dawn at extension 3191. She may be able to fix the problem, or can page Ira if necessary.

After inferring Q's plan, R evaluates it and discovers an error: Q believes that by calling Ira's office, he will contact Ira. R then decides to point that error out. Upon reflection, R notices that she has not helped Q reach his goal, so she decides to try to find another plan that Q could execute in order to get the Linc Lab laserwriter fixed. She finds such a plan, and decides to communicate the important details—that Q should contact Dawn at extension 3191.[2]

Note that the example also shows that reflection make occur over several cycles before a final response is determined. R's final statement in the example explains to Q why he should call Dawn—it explains the plan that R has decided to communicate. Computationally, reflecting upon her decision to tell Q to call Dawn, R decides that a justification of that plan is necessary.

Reflection is an important part of the cooperative response generation process. It allows CRG systems not only to explain, justify, or clarify their statements, but also allows them to perform other kinds of helpful actions such as suggesting new plans.

### 4.1.3 Summary

This section has proposed two elements of a computational model of cooperative response generation: (1) respondents must be able to select among several potential response options, and (2) CRG is partly reflective in that respondents may select new response options motivated by options already selected. An architecture that embodies these features is now described.

[2]The role of plan *synthesis* in CRG, although very interesting, is not considered in this research.

## 4.2 Top-Level System Organization

At its most general level, the CRG system architecture I propose consists of five components (cf. Figure 4.2): the *input interpreter*, the *plan evaluator*, the *response processor*, the *response generator*, and the *knowledge base*. I will briefly describe each component in the next several subsections. Following that, the response processor architecture will be discussed in detail, as it embodies the main research thrust of this work.

### 4.2.1 Input Interpreter

The user's natural language utterances are the input to this first component in the cooperative response generation "pipeline." The task of the input interpreter is to perform syntactic and semantic analysis on the input and produce as output a representation in some meaning representation language (MRL) of the literal meaning of the input. In this research, I will assume the existence of an adequate parser and semantic analyzer and take as my MRL the typed procedural calculus of Woods [60].

### 4.2.2 Plan Evaluator

The plan evaluator takes the MRL expressions produced by the input interpreter and performs the combined processes of plan inference and evaluation. Its output is a representation of the user's plan annotated with marks describing any errors that were detected (and their root causes). For simplicity, I assume that the user always provides an explicit statement of what he takes to be his goal, and that the system never encounters an ambiguous situation in which more than one possible plan can be ascribed to the user.

Because the user's goal is always explicitly stated, the output plan always contains a node reflecting that goal. The rest of the user's utterances are assumed to reflect only his beliefs about the past and present states of the world and actions performed, his desires and intentions, and his information needs. Following Pollack [47], I view plans as complex belief structures; the job of the plan evaluator, then, is to construct a representation of the questioner's beliefs, desires, and intentions[3] that accounts for all the observed utterances.

I will, however, distinguish between the questioner's *domain* plan—the plan to be used to achieve the domain goal, and his *communication* plan—the plan in execution that gave rise to the observed utterances [34].

When an agent decides that some information must be obtained in order to make further progress toward a goal, the agent forms a plan to acquire that information. Influencing this plan are the agent's beliefs about where the information is available and by what means it may be obtained. In the domain of interest here, the result of this planning process is a set of speech acts performed in an attempt to elicit the desired information from the system.

The plan evaluator will not attempt to reconstruct the questioner's communication plan. Instead, statements will be taken as assertions about the questioner's beliefs, and questions will be taken to reflect goals to acquire knowledge. The plan evaluator thus begins with a plan structure consisting of a set of isolated nodes representing the system's beliefs about the questioner's beliefs and information needs. From there, it must deduce the most plausible domain plan it can ascribe to the questioner.

[3] More precisely, a representation of the inferring agent's beliefs about the questioner's beliefs, desires, intentions, and so forth.

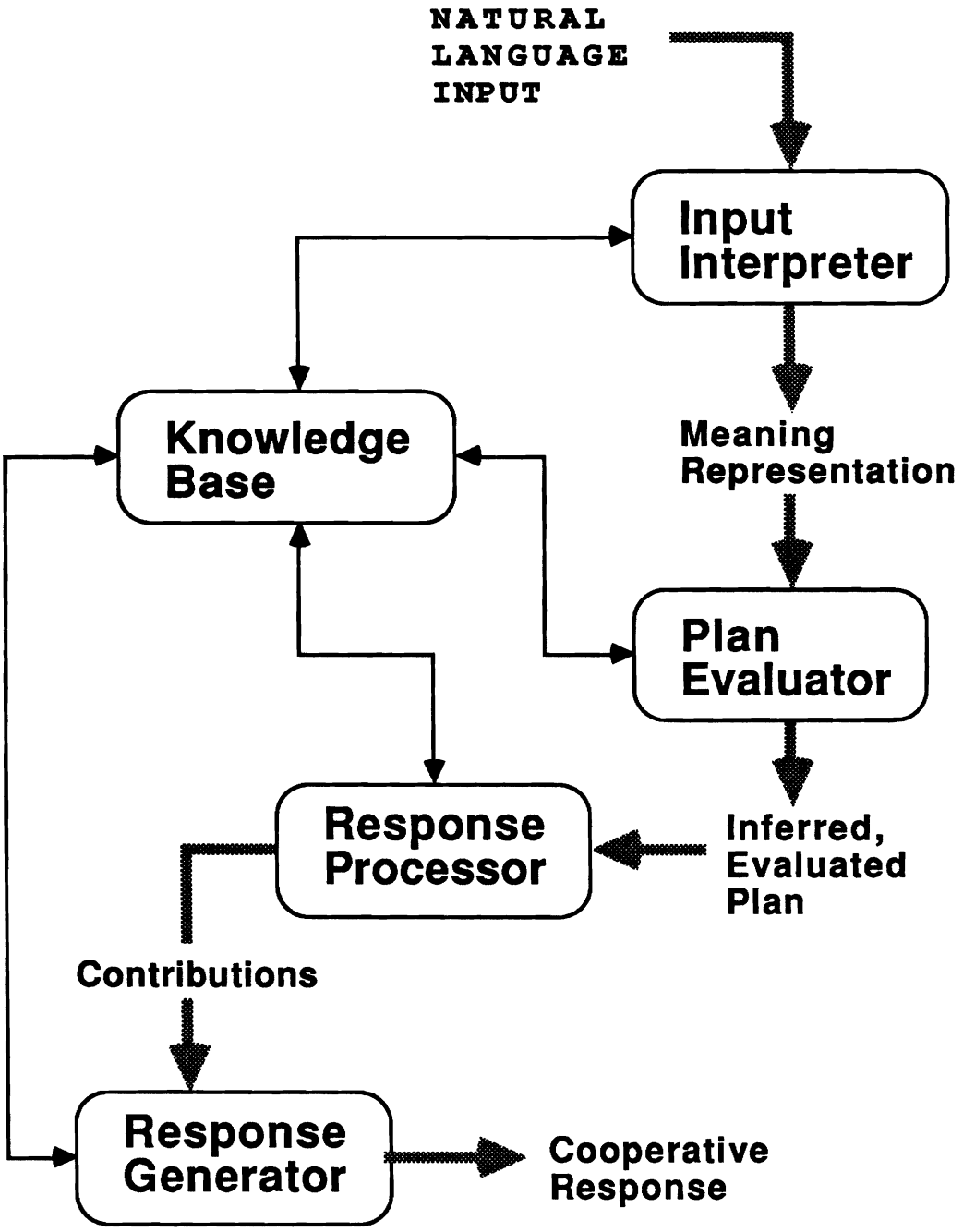



Figure 4.2: Top-Level System Architecture

### 4.2.3 Response Processor

The response processor takes the inferred, evaluated plan produced by the plan evaluator and computes a set of speech acts describing the information to be conveyed in the system's cooperative response. The response processor does not specify language structures; rather, it specifies the system's cooperative response goals. This component will be discussed in detail in Section 4.3.

### 4.2.4 Response Generator

After the response processor decides *what* the system should say, the response generator decides *how* to say it. The response generator takes the goal representation produced by the response processor and attempts to achieve those goals using language. Though other modes of communication are possible, such as speech and graphics, this research considers only text-based interaction.

### 4.2.5 Knowledge Base

Knowledge useful to than one component is maintained in a central knowledge base. This knowledge includes a domain model, a discourse model, and a user model. An effort should be made to prevent the knowledge base from becoming simply a collection of disparate databases, accessed in different ways by different modules. The abstraction of an integrated knowledge source could be maintained through the use of a common interface protocol used for all transactions.

## 4.3 Response Processing In Detail

The response processor decides what the system should say in its cooperative response; its architecture must account for the two observations discussed at the beginning of this chapter. To do so, as well as to satisfy other important design criteria, I have selected the blackboard model of problem solving as the architectural framework. The particular architecture, shown in Figure 4.3, consists of a blackboard having four sections or "tiers," and five sets of knowledge sources. The use of the tiers and the operation of the knowledge sources are discussed next.

### 4.3.1 The four-tiered blackboard

The response processor's four panels distinguish four levels of processing. The panels are labeled *Evaluated Plan*, *Meta-Response Plan*, *Response Plan*, and *Response Acts*. First the panels themselves will be discussed, then the knowledge sources (KSs) that operate on and between them.

#### Evaluated Plan

When the plan evaluator completes its work, the annotated representation of the questioner's inferred domain plan is copied to this first level of the response processor's blackboard. The representation used at this level is the one described in Chapter 3.

The plan structure itself, excluding any annotations, represents what the system believes about the questioner's beliefs relevant to the pursuit of the stated goal. The annotations highlight discrepancies between the system's and the questioner's beliefs. For example, the inferred plan might indicate that the questioner intends to call the hospital in order to contact Kathy. That is, the plan represents the configuration of the questioner's beliefs suggesting that he believes a "call hospital" action is part of a plan to talk to Kathy. In order to make sense out of this sequence, the plan

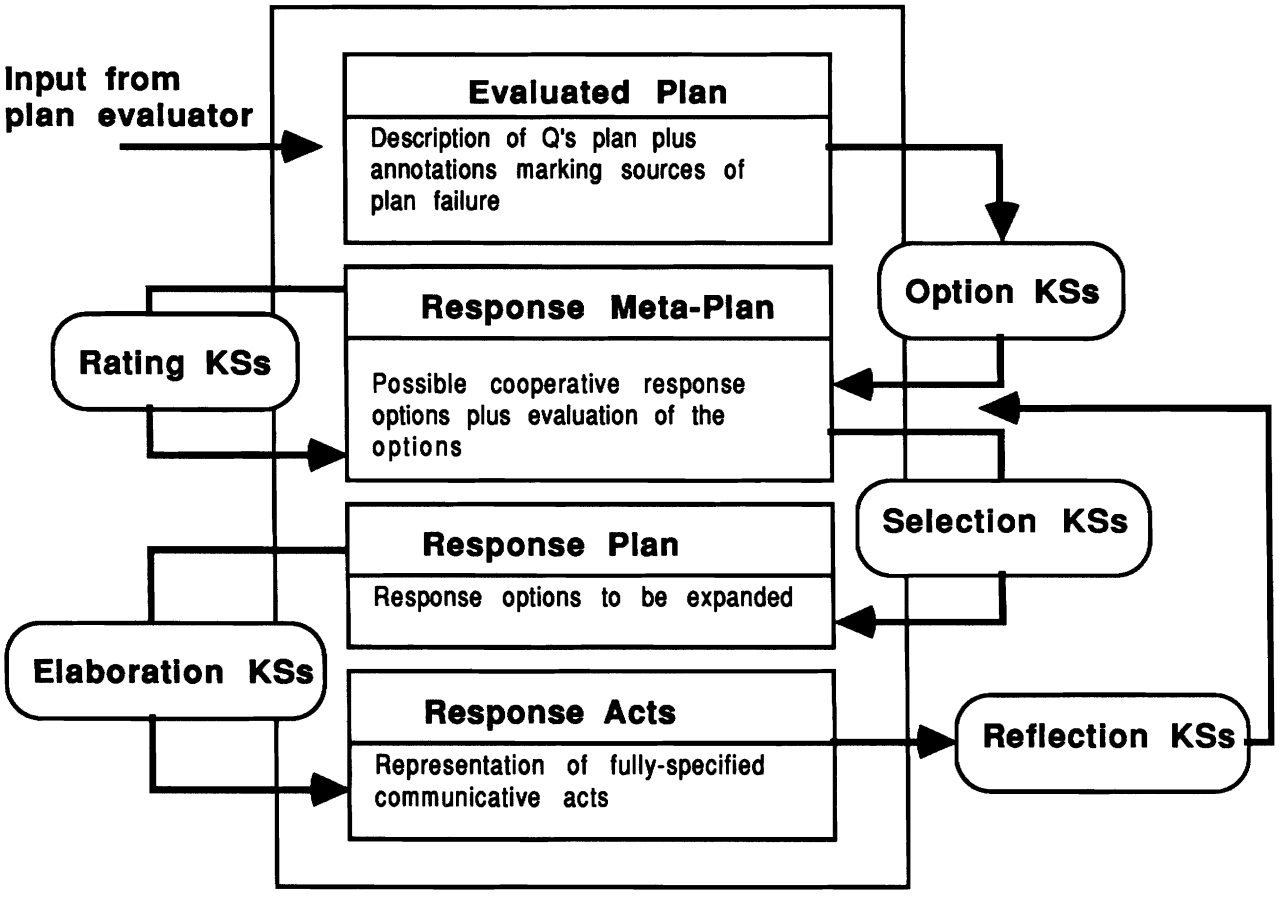



Figure 4.3: Response Processor Architecture

inference mechanism must also deduce that Q believes Kathy to be at the hospital. Plan *evaluation*, however, could reveal that the system believes Kathy is at home rather than at the hospital. The difference between inference and evaluation is that the former attempts to construct a model of the questioner's beliefs that explains the observed utterances, whereas the latter involves the comparison of the system's and questioner's beliefs.

Reasoning from the mistaken belief, the evaluator can determine that the act of calling the hospital will not contribute to the goal of establishing contact with Kathy. As I discussed in the last chapter (cf. Chapter 3, Section 3.2.4), the evaluator always marks the most specific error in a plan. Incidental effects are either omitted or clearly marked as such. Annotations will trigger KSs that suggest possible response options.

### Meta-Response Plan

My first observation on cooperative response behavior was that respondents typically exercise fewer response options than they may actually have in a given situation. They seem to use both contextual and world knowledge in order to "prune away" all but the most pertinent options. The Meta-Response Plan level of the blackboard is the first stage of the mechanism supporting that kind of behavior. Before describing this panel further, I will try to characterize the notion of "response option" more carefully.

Depending upon her interpretation of Q's request, R may find that different forms of response, indeed different response *strategies*, may be well motivated. One interpretation may be based purely on the literal meaning of the query, while others may be shaped by the context or guided by general principles of interaction. For example, the question

**Q:** What's a wheelpuller?

could be interpreted literally as a request for a definition, to which R could respond with some kind of definition for the term "wheelpuller." In a task context, where Q must carry out some action with a wheelpuller, other responses may be warranted. R could, for example, point to a wheelpuller lying on a workbench and/or say "the wheelpuller is the tool immediately to the left of the hacksaw," or simply take a wheelpuller out of a drawer and hand it to Q. Responses such as these are *contextually* motivated; it is knowledge of the context (Q's task) and an understanding of Q's corresponding needs that permit R to identify alternate possible responses.

I will label any communicative act (written, spoken, pictorial, or physical) that is motivated by a request's literal meaning, by its contextual interpretation, or by general rules of communication, a *potential response option.* That is, a potential response option is *anything* that could reasonably be said or done in response to Q's utterances (though in this work we focus only on what could be said).

The Meta-Response Plan panel of the blackboard records the currently active set of potential response options. Messages on this panel will offer different general response actions to take; these examples suggest some possible messages:

- *identify-referent* D: D is taken to be a description of some unique entity in the world, e.g., "the US Secretary of State." R may provide a more specific description of D, e.g., "George Schultz," in her response.
- *answer-yes-no* P: P is a proposition. R may answer "yes" or "no" depending upon the truth value of P.
- *define-term* T: T is a "term," a description of a concept. R may provide a definition of T (by a method to be determined later) in her response.

- *identify-unexecutable-act* A: A is an action in Q's plan that R has determined to be unexecutable (e.g., one or more preconditions will not be satisfied at execution time). R may identify A as such (using a method to be determined later) in her response.
- *correct-misconception* P: P is a proposition taken to be an erroneous belief that R ascribes to Q. This message suggests that R may attempt to correct that misconception (by a method to be determined later) in her response.

All options suggested at this level should be domain independent. At this level, the options merely detail the possibilities; they are not instantiated. There are no specifications (at this point) of how any option is to be exercised.

**Explained options** Later processes may need to know *why* response options were suggested. Knowing why a piece of information is to be conveyed will not only help the appropriate KSs rate the usefulness of the information in the response, but also will help the response generator construct readable text. Therefore each potential response option comes with an attached *explanation*: some proposition or note (exact representation to be determined) that in some way expresses the essence of why the option is considered well motivated. In the case of a *correct-misconception* P option, where P is SELLS(ACME,WINE), the explanation (for why R believes P is a misconception) could be that R believes that ACME is a SUPERMARKET, that WINE is an ALCOHOLIC-BEVERAGE, and that entities of type SUPERMARKET do not SELL entities of type ALCOHOLIC-BEVERAGE. The explanation should (concisely) capture the reasoning that led to the contribution of the option.

### Response Plan

The Meta-Response Plan level records all potential response options. At some point, a subset of those options must be selected to be part of the system's cooperative response. The selection process is the combined work of the Rating and Selection KSs, to be described shortly. Selected response options are copied from the Meta-Response Plan level to the Response Plan level. There is no change in how the options are represented.

### Response Acts

When options recorded on the Response Plan panel are elaborated (instantiated), the results are recorded on the Response Acts panel. As currently envisaged, messages at this level describe *speech acts* [3] to be performed in the response. For example, one message might be INFORM(Q,P), meaning that R should inform Q of the proposition P. Another message might be ASSERT(Q,P): assert to Q that P is true. Other speech acts include warning, requesting, promising, etc.

Elaborating a response option may result in several response acts. For example, to exercise the *correct-misconception* option, R may need to first deny the incorrect belief (assert that P is false) and then inform Q of the correct proposition.[4] So that later reasoning processes may know the origins of and reasons for individual response acts, links (pointers) are maintained between each response act (on the Response Acts panel) and its parent response option (on the Response Plan level).

[4]Note that I have here assumed that *correct-misconception* is a monolithic response option. We may want to break it apart into several more focused response options, e.g., *deny-false-belief, assert-correct-belief, justify-correct-belief,* etc. Which one is the better approach remains to be seen.

### 4.3.2 The knowledge sources

The last several sections focused on the panels of the blackboard, their meaning and use. The divisions of the blackboard distinguish the different phases of cooperative response processing. Work begins with the inferred, evaluated plan (Evaluated Plan level), from which the set of well-motivated response options are generated (Meta-Response Plan level). Next, a rating process (to be described) results in the selection of a subset of the potential response options (written on the Response Plan level), which are then elaborated (and recorded on the Response Acts level).

I have yet to show exactly how my two observations on cooperative response behavior are accounted for in this proposed architecture. The blackboard organization provides only a partial account of the first observation: the Meta-Response Plan level records all the possible response options, while the Response Plan level lists only those options chosen to be included in the response. To complete the picture and to show how reflection is accommodated, I must describe the active processing elements of the system—the knowledge sources.

There are five sets of KSs, each with a different function: the *Option* KSs, the *Rating* KSs, the *Selection* KSs, the *Elaboration* KSs, and the *Reflection* KSs. The individual KSs in the sets are all independent program modules; in keeping with the blackboard paradigm, KSs do not communicate directly with one another and cannot make any assumptions about the existence or function of other KSs. All communication is strictly indirect, occurring through messages written on the panels of the blackboard.

Each KS is an "expert" in some very specific kind of cooperative response activity. The set to which a KS belongs determines the general area of the KS's expertise. Option KSs, for example, are experts in deciding when particular response options are well motivated based on the data available on the Evaluated Plan level of the blackboard.

As currently planned, the strategy for scheduling KS execution is extremely simple: each KS, once triggered, is placed in a queue (FIFO) and executed when it reaches the front. At the moment, there does not appear to be any reason to use a "sophisticated scheduler," and there does not yet seem to be any basis for giving certain KSs priority over others. The implementation effort will shed more light on the scheduling/control problem.

The KS sets are described in the next several subsections.

#### Option KSs

Option KSs reason on the evaluated plan to generate potential response options; they read from the Evaluated Plan level of the blackboard and record their output on the Meta-Response Plan level. An Option KS is an expert in determining when a particular cooperative response option is well motivated; besides the evaluated plan, it has at its disposal all the information available in the central knowledge base.

The Option KSs become eligible to run when the Evaluated Plan level of the blackboard is updated, which in this context occurs at the completion of the plan evaluator's work. When this design is extended to cope with multi-turn discourse, it may turn out that subsequent user utterances can be used to augment or otherwise modify the initially inferred plan. Carberry has done some work in this area [5, 6].

Execution of individual Option KSs is triggered both by noteworthy configurations of user plans and by annotations generated by the plan evaluator. For example, the existence of a node N1 in a plan, linked by a *know* arc to a node N2, indicating that N1 is a *knowledge precondition* (cf. Allen [1]) of node N2, could trigger an Option KS to suggest the option of "satisfying" the user's need to know. More likely, a different Option KS would exist to suggest appropriate options for each kind

of knowledge precondition. If N1 were a KNOWREF node (indicating a need to know the referent of some description), then Option KS "KS1" might be triggered, but if it were a KNOWIF node (indicating a need to know if a particular proposition holds), then a different KS, "KS2," might be triggered instead. KS1 might suggest an *identify-referent* response option, while KS2 might suggest an *answer-yes-no* option.

Errors detected in plans also trigger Option KS activity. For example, if the plan evaluator discovers that the user intends to carry out an action when (all or some of) that action's effects will hold prior to its execution, the evaluator may annotate the node (call it N1) with an UNNECESSARY-ACTION message. An Option KS may exist with expertise in knowing how to respond when unnecessary actions are detected. That KS may recommend that the system identify the action as unnecessary, or point out the effects that already hold, do both, or do something else entirely.

In general, Option KSs may perform arbitrary reasoning in order to select response options that are well motivated on the basis of the inferred, evaluated plan. Again, it should be emphasized that the KSs do not necessarily suggest options that are appropriate or useful in the given circumstances—they merely suggest plausible options. The rating and selection process strives to extract the appropriate and useful options from the merely plausible. This is done in an attempt to reduce the amount of knowledge and reasoning required of the KSs. If each KS were required to decide not only the plausibility of options under consideration, but also their relevance, usefulness, etc., there would certainly be serious duplication of reasoning effort. In addition, many judgments of relevance and appropriateness depend on reasoning about how options interact. KSs would become extraordinarily complicated if they had to reason about all the possible interactions their options might have with other options already suggested, not to mention all the problems resulting from the introduction of processing dependencies between individual Option KSs.

By and large, Option KSs should use domain-independent rules and reasoning to identify potential response options. In particular, this should be possible because only general option *types* are being suggested, not their instantiations. It does not require any special domain knowledge to decide that an unsatisfied action precondition should be identified to the user, though it is quite possible that carrying that option out will require domain-specific reasoning.

The distinction is one of *intension* versus *extension*. Reasoning at the Meta-Response Plan and Response Plan levels is intensional—it is reasoning about descriptions of possible cooperative response acts. Much if not all of this reasoning will be independent of domain-specific rules. Reasoning at the Response Acts level is extensional, since the descriptions have been elaborated into particular instances of the intensional option descriptions. Knowing how to properly instantiate a generic response option is very likely to require specific domain knowledge.

After all triggered Option KSs have been executed, the Rating KSs are activated, beginning the rating and selection process.

### Rating KSs

Rating KSs are specialists in weighing potential response options, both individually and comparatively, and casting votes that for each suggested option will help decide whether it will be elaborated in the system's cooperative response. Rating KSs read and write on the Meta-Response Plan panel of the blackboard.

Each Rating KS carries out a single kind of test on appropriate potential response options recorded on the Meta-Response Plan panel. The tests are used to identify relevant and useful response options and to apply general principles of interaction to the process of deciding what the system should say in response. As such, though the tests themselves are focused on particular kinds

of problems, the reasoning involved may be quite general.

To illustrate, let us consider a piece of the processing of this input query:

**Q:** What's a wheelpuller?

Assume that the query is translated into a KNOWDEF (know definition) node in the inferred plan, and that contextual information has allowed the plan evaluator to construct a plan in which the KNOWDEF is a subgoal to the goal of performing some action with a wheelpuller.[5] Presumably some eager Option KS would suggest a *provide-def* (provide definition) response option. Another Option KS could very well look at the *what* question and suggest a contextually motivated response option: reasoning that Q needs a wheelpuller to satisfy his top-most goal, the KS could suggest that the system identify the particular wheelpuller lying on Q's workbench.

Under the circumstances, we probably would not want the system to define a wheelpuller as part of its response to Q's query. One basis for omitting the definition might be to compare the two options, one to define a wheelpuller and the other to identify a particular handy wheelpuller, and recognize that while both options contribute to Q's goal (after all, knowing what a wheelpuller is will help Q to find one), the latter does so more efficiently, reducing the number of steps Q must perform in order to be able to apply the wheelpuller to his current task.

Assuming such a general reasoning process could be implemented in a single Rating KS, that KS should be able to cast a vote in favor of the latter option and against the former. This brings us to the question of what exactly these "votes" are.

By default, all response options suggested by the Option KSs become part of the system's cooperative response. The Rating KSs, however, can cast "votes" for or against individual response options which in later processing determines their status, included or omitted. A vote is a message, recorded on the Meta-Response Plan panel by a Rating KS, that is either *for* or *against* a particular response option (this implies that each response option must have a unique identifier associated with it). Besides indicating *for* or *against*, each vote also contains a justification for the vote (in some suitable representation language), information that may be needed later to break ties. Tallying of votes is part of the selection process, performed by the Selection KSs, described shortly.

There will be as many Rating KSs as there are general relevance and appropriateness tests to apply. Some Rating KSs do not compare response options, but rather apply general rules to individual options. For example, suppose a KNOWREF subgoal exists in a subplan to perform an action that the evaluator has determined to be unnecessary, e.g., Q wants to know where the key to a particular door is, while R knows that the door is unlocked. Applying a general rule, a Rating KS might vote against an option to *provide-ref*, on the grounds that the information is not useful to Q's goal.

In the next section, I'll discuss how the votes are acted upon and how the Response Plan is formed.

### Selection KSs

The Selection KSs complete the mechanism that supports the first observation on cooperative response behavior. Their task is to select for elaboration response options from the Meta-Response

[5]The inferred plan might be quite involved, showing the connection between Q wanting to know the definition of a wheelpuller and Q being able to find and use a particular wheelpuller on a workbench. Of course, another possibility would be to have the plan evaluator draw on pragmatic knowledge and instead construct a plan in which the query is treated, for example, as a request to KNOWREF an available wheelpuller. This, however, seems like a hack; nor, I think, do we want to embed linguistic pragmatic inference in a plan evaluator. The query technically is a request for a definition, and the inferred plan should treat it as such. Selecting an appropriate response, one that ignores the literal meaning of the query, should be entirely at the discretion of the response processor.

Plan panel of the blackboard. They do this by tallying the votes generated by the Rating KSs; "winning" response options are simply copied to the Response Plan panel.

How are "winners" determined? As currently planned, each vote has equal weight, and winners are chosen by simple majority: a response option is selected if it has more FOR votes than AGAINST. A response option without votes either way is selected by default.

In the event of a tie, special Selection KSs may be called upon to break the tie based on knowledge derived from the justifications of the votes. It is also possible that a simpler strategy will be sufficient; for example, satisfactory behavior may be obtained if ties are routinely treated as wins (or losses). The implementation effort should yield useful information in this area.

Once response options have been selected for the Response Plan, they must be elaborated, the task of the next set of KSs.

### Elaboration KSs

The processing thus far, from the Evaluated Plan to the Response Plan level, has been concerned with selecting a set of general actions for the system to take in its response. For example, the system might decide both to answer Q's *wh* question and provide some additional information believed useful to Q.

The response options selected and recorded on the Response Plan panel of the blackboard may be viewed as a set of goals that the system wants to achieve in its response. The work of the Elaboration KSs is to instantiate a plan, in this case taking the form of a set of speech acts, that the response generator can use to construct text that will achieve those goals. The Elaboration KSs record their results on the Response Acts panel of the blackboard.

For each general response option that may be suggested by an Option KS, there must be at least one Elaboration KS that knows how to instantiate it. The elaboration process is likely to involve quite a lot of computation since that is when the actual work of putting the response together takes place. For example, if *correct-misconception* is part of the Response Plan, then a mechanism such as that developed by McCoy [37] may need to be invoked to compute how the correction is to be effected. Elaborating an *identify-referent* is likely to require one or more knowledge base references and associated computations.

Elaboration KSs compute representations of the speech acts to be performed by the system in its response. These are taken by the Response Generator as directives, guiding its choice of lexical items and grammatical structure in order to achieve the desired illocutionary and perlocutionary effects.

The representation will consist of speech act names as functors with arguments represented in a predicate calculus. Thus INFORM(P) indicates that the response should inform Q that the proposition P holds. All literals in P will denote knowledge base entities; the Response Generator must choose appropriate forms of reference, be they noun phrases or anaphors. The response generator actually has a fair amount of latitude in deciding how to best phrase the system's reply; the contents of the Response Acts panel comprise its particular communicative goals.

In contrast with the way response options are selected, response acts do not go through a phase of rating and selection once generated. Though implementation may suggest that such a facility is needed, at present the goal is to ensure that response options are narrowly enough defined so that only one elaboration will be appropriate in any given situation.

**Reflection KSs**

The Reflection KSs give the system the ability to augment its response (prior to response generation) in order to explain, justify, qualify, elaborate, or otherwise accommodate for information conveyed by existing response acts. These KSs analyze records on the Response Acts panel of the blackboard and may introduce new potential response options on the Meta-Response Plan panel. New potential response options are then processed in the normal way; Rating KSs vote on their relevance and appropriateness and Selection KSs decide whether to make them part of the Response Plan and hence eligible for elaboration.

Reflection KSs are needed to perform all cooperative response reasoning that depends on information available only in elaborated response options. For example, Joshi's addendum to Grice's Maxim of Quality admonishes a respondent to explicitly modify her reply to block any potential misconstruals she detects [26]. This behavior would be properly implemented in one or more Reflection KSs. Such KSs could inspect already-selected response acts to see if the information they convey licenses any false inferences.[6] If so, appropriate new response options could be generated. Potential response options generated by Reflection KSs would have justification links pointing to the parent response acts recorded on the Response Acts blackboard panel.

As another example, the behavior of Kaplan's COOP [31] system might be completely implemented by one or more Reflection KSs. Given a query like "How many French students passed CIS577 in Spring of 1988," and assuming no plan-related knowledge suggests that a direct answer is inappropriate, an Option KS might suggest a *direct-answer* option (meaning to retrieve the answer from the knowledge base and include it in the response). If selected and elaborated, the computed response act might be to ASSERT that no French students passed the designated course. A Reflection KS, noting the null result, might test various sub-descriptions of "French students who took CIS577 in Spring 1988." If it found a failed sub-description, the KS could generate a new option suggesting that the system identify the part of the query that has a null extension.

In principle, Reflection KSs could introduce response options that subsequently trigger other Reflection KSs. Thus some mechanism may be needed to ensure that response processing does not iterate endlessly.

The next section considers in detail three variations on one question-answer exchange, illustrating how a system designed according to the proposed architecture might compute the cooperative response.

## 4.4 A Processing Example: Variations on a Theme

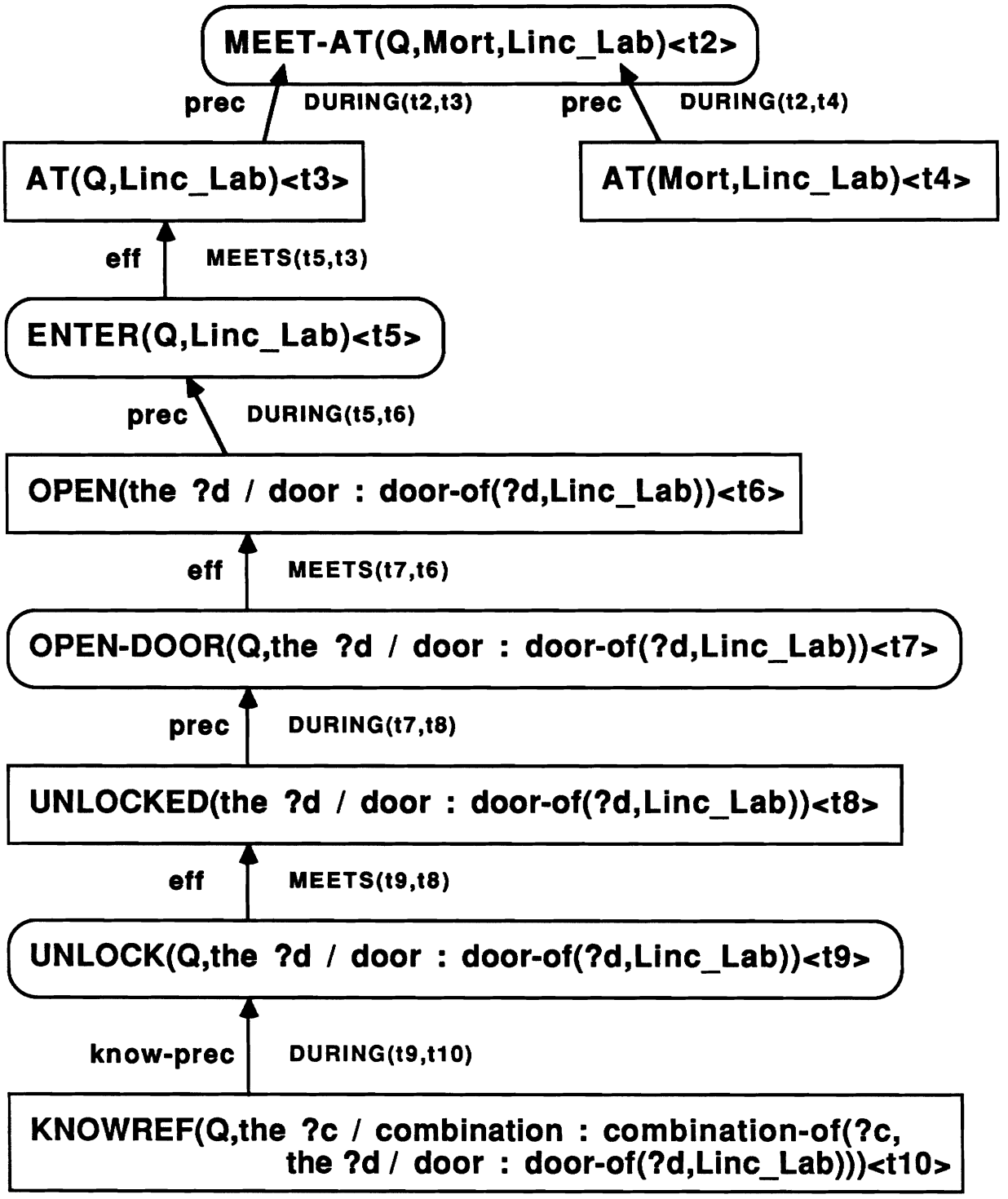
In the three variations to be considered here, the question will be the same:

> **Q:** I want to talk to Mort. What's the combination to the Linc Lab door?

The differences in response will hinge on differences in R's beliefs about the state of the domain. In all examples, the inferred plan will be as shown in Figure 4.4.

The inferred plan represents R's beliefs that Q wants to know the combination to the Linc Lab door in order to unlock it and open it, ultimately enabling Q to enter the Linc Lab and meet Mort. The links are labeled, and for clarity I have also marked the links with the temporal relationships. The annotations produced by the Plan Evaluator will vary depending on the situation.

[6] Inferences, false or not, may be licensed by the *content* of what is said or by the *way* that content is expressed. In its present configuration, the architecture provides no way for the Reflection KSs to reflect on the response generator's lexical and grammatical choices.




---

```
SBQB(HOLDS(LOCKED(the ?d / door : door-of(?d,Linc_Lab)),t11)

MEETS(t11,t9)
```

Figure 4.4: Inferred Domain Plan

### 4.4.1 Example 1

The first example will be the simplest. Assume a scenario in which R believes the door is locked and Mort is in the Linc Lab. In this case, a simple reply would suffice:

**R:** It's 5421.

That is, R just provides the direct answer.

The Plan Evaluator, finding Q's plan to be perfectly reasonable, produces no annotations to the inferred plan. The unmarked plan is then copied to the Evaluated Plan panel of the Response Processor's blackboard.

#### Step 1: Option KSs

The inferred plan I have shown treats Q's question as a KNOWREF request, that is, a request to be informed of the referent of the description "the combination to the Linc Lab door." The question could also be taken as a KNOWDEF, a request to know the *definition* of the concept "the combination to the Linc Lab door." I expect that the plan inference machinery, in its attempt to find a plan that explains how wanting to know the combination could connect with meeting Mort, should be able to discard the second interpretation. In any case, the availability of a second interpretation would not present any difficulty, although more processing would be required to obtain the desired behavior.

Because the KNOWREF node represents a request by Q, it requires special treatment; there should always be an Option KS to suggest response options appropriate to a direct request. In this example, an Option KS could suggest that R exercise the *provide-referent* response option. A potential response option record would be written on the Meta-Response Plan level of the blackboard, with an explanation indicating that Q requested to know the referent of "the combination to the Linc Lab door."

If the alternate interpretation of Q's question were available, a different Option KS might suggest a *provide-definition* option, explaining that Q requested the definition of "the combination to the Linc Lab door."

No other obvious response options suggest themselves; for simplicity let us assume that no other Option KSs have any contributions to make.

#### Step 2: Rating KSs

The Rating KSs look for reasons to vote for or against suggested response options. One circumstance in which a Rating KS might vote *for* a potential response option is when an independent reason (separate from the original motivation) is found to motivate exercising that option in the response. The effect of this is to promote response options that satisfy more than one goal. In the example at hand, there do not appear to be any independent motivations for carrying out the *provide-referent* option.

Rating KSs vote against a response option when they find reasons to suggest that the option is irrelevant or inappropriate. In the case of *provide-referent*, no arguments should be found against it.

If the *provide-definition* option had been suggested, then at least one Rating KS should find grounds to vote against it. Given my working definition of a cooperative response as one that is both informative and non-misleading, one (or possibly more than one) Rating KS should be designed to rate the informativeness of response options. Informative responses provide new information; one must not tell people things that one believes they already know. If a user model were available,

a Rating KS might use it to evaluate whether defining "the combination to the Linc Lab door" would be informative, for example, by testing if the user model indicated that the questioner already understood the concept. Without a user model, a Rating KS could test if definitional knowledge was useful in reaching the top-most goal.

Although it is quite possible to build Rating KSs that perform very specific (and perhaps *ad hoc*) tests, doing so would violate the spirit of the architecture design. Rather, Rating KSs are intended to provide a vehicle with which to design and test general principles of cooperative interaction.

Returning to the example, if only the *provide-referent* option was suggested, there would be no Rating KSs voting either for or against it. If the *provide-definition* option was suggested, at least one Rating KS should carry out the appropriate test and vote against it.

### Step 3: Selection KSs

As soon as there are no Rating KSs waiting to execute, the Selection KSs would begin to inspect the Meta-Response Plan panel of the blackboard. In this example, that panel would contain either the *provide-referent* option only, or it might contain both the *provide-referent* and *provide-definition* options, plus any votes recorded against the latter option.

We will assume that selection is by simple majority, with the default being to select. Thus, the *provide-referent* option would be selected by default (no votes for or against). The *provide-definition* option would not be selected due to the majority of votes against it. At the end of selection, the *provide-referent* option record would be copied to the Response Plan panel of the blackboard.

### Step 4: Elaboration KSs

There should be one Elaboration KS that knows how to instantiate *provide-referent* response options. One of the first tasks of that KS is to find out from the knowledge base exactly what the referent of "the combination of the Linc Lab door" is. That may turn out to be some knowledge base entity, say of type LOCK-COMBINATION, denoting the numeric combination "1234."

Once the KS has found the internal entity corresponding to the given description, it must construct a speech act representation for use by the response generator. That is, it must create a message for the response generator telling it what communicative act should be performed and what information that act should convey. In this case, the KS might produce a message saying, in effect, "inform the user that G1432 is the referent of the description 'combination to the Linc Lab door'," where G1432 is the internal knowledge base entity satisfying the description.

The response generator selects the words and structures to be used to convey the desired information. It decides how to describe knowledge base entities to the user. At present, I hope to be able to maintain a clean separation between the work of the Elaboration KSs and that of the response generator. The implementation effort should yield some useful insight into how much information the Elaboration KSs need to provide in order for the response generator to perform successfully.

### Step 5: Reflection KSs

In this simple example, there is no obvious need for reflective response processing. After all Elaboration KSs have run, the results on the Response Acts panel of the blackboard could be passed on to the response generator. However, to provide some insight into how reflective processing might work, let's suppose that the system does not know what the combination to the Linc Lab door

is. That is, suppose that the Elaboration KS attempting to realize the *identify-referent* option discovers that the information isn't in the system's knowledge base.

There are at least a couple of ways in which this situation could be handled. One way would be to have the plan evaluator, as part of its normal activities, look up the referents of all embedded descriptions. Then the error could be discovered early and dealt with in the normal way, with Option KSs suggesting possible relevant options, and so forth. However, looking up referents involves one or more knowledge base transactions and is potentially expensive. Furthermore, later processing may not need those referents looked up. Thus, it seems that deferring such operations until they are demanded is a reasonable policy.

The other approach is to deal with such failures using the reflection mechanism. The Elaboration KS handling *identify-referent* might, upon failure, post some kind of descriptive error message (not to be taken as an actual response act) on the Response Acts panel of the blackboard. That error message could in turn trigger a Reflection KS, which might suggest a new response option, say *answer-unknown*, by which the system would indicate it did not have the requested information.

When all Reflection KSs are finished, there may be new response options on the Meta-Response Plan panel, in addition to those already there. At this point, Rating KSs become active once more. The presence of the new options could conceivably cause previously untriggered Rating KSs to vote for or against options that had been on the Meta-Response Plan panel before. Votes, once cast, are never erased; new votes are simply added to the collection. As a result, there is no guarantee that processing will not be expended needlessly. As a result of reflection, new votes could oust previously selected options.

At the end of the rating process, the Response Plan panel would be erased, and a new set of options selected based on the most recent ratings. Elaboration would take place again, perhaps followed by further reflection, until iteration ceased. Assuming only one cycle of reflection, we see how this mechanism could allow the system to answer "I don't know" if that information was not in the knowledge base.

**Summary**

This example shows how a simple question, one involving no special processing in order to be cooperative, could be answered by a system having the NLQA architecture I propose. The next example shows how a cooperative response could be generated in cases where a direct answer is inappropriate.

### 4.4.2 Example 2

The second example involves a slightly modified version of the Example 1 scenario. This time, assume R believes the door is unlocked. Then this would be an adequate response:

**R:** It's not locked.

It might also be reasonable to include the direct answer as well:

**R:** The combination is 5421, but the door is unlocked.

As I will show, the architecture permits either response; the system developers must devise and implement the principles that guide the preference of one response over another.

In this case, the Plan Evaluator should detect that Q intends to perform an action (UNLOCK) to achieve an effect (the Linc Lab door being unlocked) that is expected to hold regardless of the execution of the action. An annotation to that effect should therefore be placed on the Evaluated

Plan panel of the blackboard, along with the representation of the plan itself. Processing begins with activation of the Option KSs.

### Step 1: Option KSs

Once again, the *provide-referent* option is suggested, motivated by Q's request (let us no longer consider the *provide-definition* option). The presence of the Plan Evaluator's annotation sparks two other options:

- *identify-unnecessary-action*: point out that the UNLOCK action is not necessary.
- *identify-S-bel-effect-holds*: point out that the system believes that the effect of the UNLOCK action already holds.

Both options would be justified by the presence of the Plan Evaluator's annotation to the inferred plan. Each option, however, represents a different possible approach to bringing the failure to the user's attention.

### Step 2: Rating KSs

As with the first example, no independent motivations are found for the *provide-referent* option. However, depending upon the behavior desired by the system developers, reasons may be found to vote against that option. For example, the KNOWREF subgoal may be viewed as part of what the system believes to be an unnecessary subplan. That is, if Q does not need to unlock the door, he does not need to know its combination. Computationally, a Rating KS could be built that generated votes against response options when they pertain to plan nodes that are within subtrees of unnecessary actions. Since the *provide-referent* option has no independent votes in its favor, a single vote against it would be enough to keep it out of the system's complete response.

Again, depending on the behavior desired by the system developers, either of the options *identify-unnecessary-action* or *identify-S-bel-effect-holds* can be included in the system's response based on how Rating KSs are designed to vote. The behavior I envisage would suppress the former option in favor of the latter. The driving principle would involve the idea that R, by suggesting that an action's effect will already hold, implies that the action is not needed. A Rating KS that could evaluate options according to that principle might do two things: vote for the option that implies the other, and against the option so implied. As a result, it would take two votes against *identify-S-bel-effect-holds* to suppress it (assuming a tie means select) and a vote for *identify-unnecessary-action* to make it once again eligible for selection.

### Step 3: Selection KSs

Selection works as before. Depending on the desired behavior, controlled by the Rating KSs, *identify-referent* might be selected together with *identify-S-bel-effect-holds.*

### Step 4: Elaboration KSs

The Elaboration KS for *identify-S-bel-effect-holds* must find a way to realize that option. There are two ways in which it might be accomplished: R could simply inform Q that the door is currently unlocked, or R could tell Q that the negation of the effect does not hold—the door is not locked. That choice is up to the KS; the decision could be made arbitrarily, or some reason might be found to prefer one to the other.

Assuming that no need for reflection arises, the Response Acts panel would be left with either one or two messages on it. The elaboration of *identify-S-bel-effect-holds* would certainly appear; the elaboration of *identify-referent* would optionally appear.

#### Summary

This example showed how an irrelevant direct answer could be suppressed in favor of (in a sense) correcting a misconception. We begin to see that a great deal of responsibility for ensuring a coherent response lies with the Rating KSs.

### 4.4.3 Example 3

This last example assumes a scenario in which R believes (1) the door is unlocked, and (2) Mort is not in the Linc Lab. In this case, either of the following responses might be desirable:

**R1:** Mort is not in the Linc Lab.
**R2:** The door is unlocked, but Mort is not in the Linc Lab.

As before, the Plan Evaluator should mark the plan indicating that the UNLOCK action is unnecessary. This time, however, it should also indicate that the system and user disagree on Mort's location: for Q's plan to make sense, he must believe Mort is in the Linc Lab, contrary to what the system believes.

#### Step 1: Option KSs

Once again, *identify-referent* should be suggested, triggered by the presence of the KNOWREF node. As in Example 2, both the *identify-unnecessary-action* and *identify-S-bel-effect-holds* options should be suggested as well. The new annotation should spark options such as these:

- *identify-false-belief*: point out that Q's belief that Mort is in the Linc Lab does not hold.
- *identify-plan-failure*: indicate that the (inferred) plan will not successfully achieve its goal.

That is, seeing that the plan won't work, one possible response is to say as much, while another is to identify what one believes the problem to be.

#### Step 2: Rating KSs

The *identify-referent* should be handled as it was in earlier examples. Let us assume that a Rating KS casts a vote against it because it is part of an unnecessary subplan (the subplan to unlock the Linc Lab door). The options provoked by the unnecessary UNLOCK action are mostly handled as before; because the information conveyed by *identify-unnecessary-action* is implied by *identify-S-bel-effect-holds*, a vote is cast against it, while a vote is cast for *identify-S-bel-effect-holds*. Note, however, that similar to *identify-referent*, both options are themselves now part of an unnecessary (invalid) plan. The fact that the plan as a whole is believed to be flawed (because of Mort's absence) argues against options concerning its subparts. So the same Rating KS that voted against *identify-referent* could also vote against *identify-unnecessary-action* and *identify-S-bel-effect-holds*.

At this point, *identify-referent* would have one vote against, *identify-unnecessary-action* would have two votes against, and *identify-S-bel-effect-holds* would have one vote for and one vote against. So *identify-S-bel-effect-holds* remains eligible for selection. If a vote had not been cast in its favor, it would not be eligible.

Options *identify-false-belief* and *identify-plan-failure* are handled like *identify-unnecessary-action* and *identify-S-bel-effect-holds*, with *identify-false-belief* favored because it implicitly conveys that the user's plan is flawed.

#### Step 3: Selection KSs

Assuming the voting scheme just described, the options *identify-false-belief* and *identify-S-bel-effect-holds* would be selected and copied to the Response Plan panel of the blackboard.

#### Step 4: Elaboration KSs

Elaboration of *identify-false-belief* should yield a directive to inform Q that the system does not believe Mort is in the Linc Lab. As before, *identify-S-bel-effect-holds* is realized by asserting that the Linc Lab door is unlocked.

One of the sample NL responses was this:

**R:** The door is unlocked, but Mort is not in the Linc Lab.

It remains unclear exactly how the Plan Generator decides to order the assertions in this way and connect them with a "but." Some work must be done to find out how much information must be passed to the generator in order to make such decisions possible.

#### Step 5: Reflection KSs

The *identify-false-belief* option, when elaborated, presents an opportunity for reflective response processing. If the system is going to tell Q that some proposition $p$ does not hold (where $p$ in this case is "Mort is in the Linc Lab"), it may be able to find some *related* proposition $p'$ that is true and would be of interest to Q. For example, say that Mort is actually somewhere else, like in Ira's office (this information might be discovered as a by-product of plan evaluation). A Reflection KS might suggest telling Q that. If that new option survived selection, then perhaps the following more complete response would be possible:

**R:** The door is unlocked, but Mort is not in the Linc Lab, he's in Ira's office.

Of course, this leaves open the question of how one goes about finding useful related propositions, but that is a separate research issue.

## 4.5 Concluding Remarks

In this chapter I have presented and discussed an architecture for cooperative NLQA systems. I began by making two observations about cooperative response behavior that I believe must be accounted for in an architecture design. The architecture I went on to propose accounts for both phenomena, the first by the rating/selection process, and the second by the Reflection KSs.

I used the blackboard model as the foundation for my architecture design. Besides the theoretical motivations for that design, the blackboard approach also has an important *practical* benefit: blackboard systems are highly modular. Why should modularity be a design consideration?

Since CNLQA systems are going to be large software artifacts, their architectures should be designed with *modifiability* in mind. A modifiable CNLQA system should have the following properties:

- The range of the system's abilities should not be fixed by the architecture. Rather, it should be possible to extend the system over time, adding new forms of cooperative behavior to its repertoire as they are studied and implemented.
- Adding new forms of cooperative behavior should have minimal impact on those parts of the system already in place and functioning.
- Once a particular form of behavior has been incorporated into a system, it should be possible at any later date to remove, replace, or upgrade it with minimal effect on the rest of the system.

Modular systems are modifiable. In a blackboard system, knowledge sources operate independently, communicating only indirectly by means of the blackboard. As long as the KSs are not allowed to make any assumptions about the existence or function of other KSs, it should be possible to build systems that allow individual KSs to be added, removed, repaired, or replaced, without requiring changes to other parts of the program. Besides being well-suited to the CRG task, the blackboard architecture is likely to both simplify and accelerate the implementation of CNLQA systems, as well as ease the work of system maintenance.

There is, of course, much work to be done in order to evaluate the usefulness of the proposed architecture. It is clear that the Rating KSs have a great deal of authority and responsibility. Some effort should be spent to determine if some kind of minimal set of general Rating KSs can be found. The vote-tallying process will require testing in the implementation; it may be necessary to consider weighted votes.

As it stands, the architecture proposed here is a simple framework for implementing cooperative NLQA systems. Many refinements are possible; an extended implementation effort will be useful in deciding which modifications are useful. The next chapter discusses my plans for carrying out the research in the areas that I think are crucial to validating the approach as a whole.

# Chapter 5

# Proposal for Research

The preceding chapters have discussed the motivation for and design of an architecture for cooperative natural language question-answering systems. As I stated in the introduction, this research aims to demonstrate the validity of two claims:

1. plan evaluation is an important step in selecting a cooperative response, and
2. cooperative response generation can be computationally modeled using a constrained blackboard architecture.

In order to prove these claims, a prototype cooperative NLQA system must be designed, implemented, and analyzed. This chapter describes my plans and goals for the implementation effort and reviews some questions that I hope will be answered in the process.

## 5.1 Implementing a Cooperative NLQA System

In artificial intelligence research, perhaps more so than in other branches of computer science, the proof of the claim is in its implementation. Formalizations that look plausible on paper often turn out to be practical failures when the time comes to realize them on a computer. Implementation ferrets out hidden assumptions upon which a theory rests, assumptions too easily introduced when the object of study is intelligence itself. Strong theories withstand their assumptions.

My implementation has two claims to prove. In this section, I will suggest how an implementation might do so. I will consider each claim and describe what demonstrable features of an implementation could be taken as its proof. The section concludes with an outline of the particular prototype system I intend to build.

### 5.1.1 Proving the claims

To prove that plan evaluation is useful in selecting a cooperative response, an implementation must be shown to use the output of a plan evaluator in its reasoning leading to a reply. This could be accomplished by showing a system whose response behavior varied according to the kinds of errors detected by the plan evaluation component. For example, run the system several times on the same input data, each time changing only the system's beliefs about the world or the domain so as to cause different errors to be detected in the user's plan. Assuming that the computational mechanisms involved are free of special accommodations for the particular input, the claim is proved if the system's response behavior differs appropriately in each case.

This query, discussed at length in the last chapter, would make an ideal test case for an implementation:

**Q:** I want to talk to Mort. What's the combination to the Linc Lab door?

A program that could produce appropriate cooperative responses in the three situations discussed, using no *ad hoc* machinery, would constitute a proof of the claim. Of course, proving the claim does not imply that it's a useful result. That argument is strengthened according to the breadth of examples that can be accounted for and handled computationally with the help of plan evaluation.

Proving the second claim is a much larger task than proving the first. The simplest proof would be an implementation that has the architecture I propose and is able to produce responses satisfying my definition of "cooperative." But the question of whether a system having a constrained blackboard architecture *can* produce cooperative responses is not nearly as interesting as the question of how *useful* that approach is in general.

Many important questions were raised in Chapter 4, all of which should be answered by an implementation. For example, a consistent methodology for defining general response options is needed. As I develop the prototype, a decomposition of cooperative responses into response options will necessarily occur. When the programming is concluded, the particular partitioning that I used should be evaluated.

The rating and selection process must become better understood. Proper system behavior depends heavily on proper rating and selection of response options. It must be ascertained that the reasoning abilities required of the Rating KSs are constrained enough to be implementable. Ideally, some small and general set of rating procedures can be identified.

The distinction between reflective and non-reflective response behavior is not always clear. The implementation effort should either provide a basis for distinguishing the two, or suggest a better generalization to replace reflection.

Other more general questions should be answered through implementation. The response processor is claimed to be modular and hence modifiable. How does that work out in practice? How easy does it turn out to be to integrate different kinds of cooperative behavior in this framework?

The representation of blackboard messages has been all but ignored. Clearly, this too must be dealt with; the needs of the evolving system will guide the design of the internal representation.

### 5.1.2 Test suite

Several examples, if reproduced computationally, highlight the features and advantages of the proposed architecture. The example from Chapter 4 is particularly good in this respect. Given this query:

**Q:** I want to talk to Mort. What's the combination to the Linc Lab door?

the system should be able to produce any of these responses:

**R1:** It's 1234.
**R2:** The door isn't locked.
**R3:** Mort is not in the Linc Lab.
**R4:** Mort is not in the Linc Lab, he's in Ira's office.

Differences in response behavior will depend upon the system's beliefs about the status of the door and Mort's location. Besides demonstrating the usefulness of plan evaluation, this example exercises the Rating and Selection KSs. When more than one error is found, rating and selection can be designed to produce sensible combinations of response behavior, e.g., conveying both R2 and R3:

**R5:** The door is unlocked, but Mort is not in the Linc Lab.

To implement this example, I will have to develop a methodology for identifying and implementing different general kinds of response options, and then flesh out the rating and selection process. Implementing this example will also provide an opportunity to design a general Rating KS—the KS that rates response options based on their pertinence to valid parts of the questioner's plan.

Although it's an interesting subject for study, I won't consider how plan generation (synthesis) is used in cooperative response generation. In particular, I will ignore responses that depend on the system computing a plan that achieves Q's goal.

This is another interesting example worth working on, interesting in part because it's a variation of Grice's classic example:

**Q:** I want to buy some wine. Where is the nearest supermarket?

In a domain (such as Pennsylvania) where supermarkets do not sell wine, the system should correct Q's misconception:

**R:** You cannot buy wine at supermarkets, only at state liquor stores.

Mays has labeled the error underlying Q's query an *intensional query failure* [35]. That is, at the level of domain description, "supermarket" objects are not characterized as "selling" objects of type "wine" (this is in contrast with an *extensional query failure*, in which a description fails to denote any object in the domain).

The response denies Q's false belief—that wine is sold at supermarkets—and justifies that denial by explaining where wine *is* sold. A more helpful response might include additional information of this form:

**R:** The nearest state liquor store is at 40th and Market.

A "helpful addition" such as this should be handled using the reflection mechanism. The addition is motivated not by what Q has said or requested, but by R's recognition that Q's goal—to buy wine—is not furthered satisfied by the denial and justification. Reflection could lead R to search for ways to help Q despite the errors detected in his plan.

Although plan inference and evaluation is not needed to do so, the system should be able to provide simple non-misleading answers to direct questions such as the following, especially as they may involve reflection:

**Q:** Which 24-hour drug stores sell milk?
**R1:** 17 *vs.* All of them.
**R2:** None *vs.* There are no 24-hour drug stores.

Another set of examples worth implementing in this framework is taken from Pollack's dissertation:

**Q:** I want to prevent Tom from reading my files. How do I set the file permissions to faculty read-only?
**R1:** Type "set perm read=faculty."
**R2:** Tom is a member of the faculty.
**R3:** There is no such file permission as faculty read-only.
**R4:** Tom is the system manager and cannot be prevented from reading your files.

It will be of particular interest to see how easily the kinds of behavior that SPIRIT was designed to produce can be integrated with the kinds of behavior demonstrated in the other examples.

Finally, it would be enlightening to try to reproduce computationally the kinds of exchanges seen in the Berkeley transcripts, modified to fit our restrictions on input structure. For example:

**Q:** How do I send mail to ucb-ernie? Address y:cc-48 didn't work.
**R1:** The correct address syntax is cc-48@ucb-ernie.
**R2:** Ucb-ernie is no longer on the network. All of ucb-ernie's users have been moved to ucb-fred. The proper address is now z:cc-48.

### 5.1.3 Plans for implementation

I have described the implementation in terms of the demonstrable features that would count as successful proof of the claims of this thesis, and have listed several examples that the prototype system should be able to handle. Now I will present some technical details about the implementation as currently envisaged.

All domain knowledge will be represented using the KL-TWO knowledge representation system [53]. KL-TWO combines a terminological (intensional) representation component with an assertional (extensional) reasoning component. Domain entities, their characteristics and so forth, are defined by positioning them in a taxonomy of other domain entities. Instances of domain entities, for example, a particular Thriftway supermarket in Philadelphia, are represented and manipulated by the assertional component.

KL-TWO provides a set of primitives that allow domain concepts and instances to be defined and reasoned about. A year or so ago, I wrote a set of interface routines to hide the details of KL-TWO from a user or system, specifically, the differences in handling terminological and assertional knowledge. Part of this package allowed simple MRL-style descriptions of things like "the combination to the Linc Lab door" to be evaluated or tested against the knowledge base. This package needs to be revised and extended to meet the needs of processing some of the more complicated examples in the test suite.

Unless I am able to import an adequate parser/semantic interpreter and get someone to set it up properly, I will simply hand-code the representations of the input utterances in a MRL-style language.

I will attempt to implement a simple plan inference/evaluation component. Inference rules will be along the lines of Allen's, modified to accommodate temporal reasoning. I will see how far I can get without having to assume a user model. As a simplification, I will assume that only one plausible plan is ever inferred. This effort should take approximately four months to complete, including start-up time on the Symbolics Lisp Machines and time spent getting the knowledge representation tools working and building a testbed domain model and knowledge base.

I will implement the response processor at all levels necessary to handle the examples discussed in the last section. During this work, I will focus on techniques for identifying and implementing response options, then representation of blackboard messages, and different rating and selection strategies. I will investigate the problem of ensuring that reflection always terminates. Elaboration of response options will be kept simple, since determining proper elaboration schemes for different response options may be research projects themselves. I'm expecting this effort to take about six months.

Implementation of the response generator will be largely ignored. If time permits and the job seems feasible in the small scale, I will build a simple template-filling generator, one that might not produce particularly elegant natural language text, but one that would at least suggest that, given

a more sophisticated generator, the complete system could produce reasonable responses. Lacking a generator, I will argue that the information content selected by the response processor, if realized in natural language, would constitute cooperative response behavior.

## 5.2 Concluding Remarks

In this thesis, I developed a model of cooperative response generation and proposed an architecture for cooperative natural language question-answering systems. I claimed that *plan evaluation* should be considered an integral reasoning step in the selection of a cooperative response. The proposed architecture, based on the blackboard model of problem solving, supports the design of systems able to do what no system could do heretofore: integrate arbitrarily many forms of cooperative response behavior and reasoning.

At the very beginning of this report, I suggested that the results of this research are applicable beyond natural language systems. This thesis aims to model the mechanisms underlying cooperative interaction; written natural language simply provides a convenient mode of communication to study. The system could just as easily get its input data by speech or vision. No matter how the input is obtained, the core reasoning steps remain the same: the user's plan must be identified and evaluated, then the respondent must consider the possible response options and select the information to convey (or actions to perform) in reply that is most appropriate in the given situation.

One reasoning process that has gone unmentioned is *plan synthesis*. Many examples in the transcripts I studied suggest that respondents occasionally generate new plans for their questioners and attempt to communicate the important details of those plans in their responses. Plan synthesis is usually invoked when the respondent believes that the questioner's plan is either totally unworkable or inefficient. Incorporating plan synthesis into a model of cooperative response generation will be left as an open research issue; however, it seems that the architecture I have proposed is sufficiently extensible. Plan synthesis can be invoked as a new reasoning component prior to response processing, or it could be distributed as necessary in Option and/or Reflection knowledge sources.

**Acknowledgements** My advisor, Bonnie Webber, deserves my special thanks for her unwavering support, encouragement, and trenchant commentary throughout the development of this thesis. Mitch Marcus was of particular assistance in getting this project going on the right track. My indebtedness to Aravind Joshi in all matters is beyond calculation. Ellen Hays, Robert Rubinoff [50], Marie Macaisa, and Bob Kass all provided many hours of thoughtful discussion on the topics addressed herein. I thank my entire committee, James Allen, Norm Badler, Aravind Joshi, and Mitch Marcus, for helping me focus this work; their comments so many moons ago have, I think, vastly improved the coherence of this research. Finally, without manta, my unix-pc at home, the completion of this document might have taken three times longer. Of course, any and all shortcomings of this work are entirely my own fault—except for typos, which manta should have caught.

# Bibliography


[1] James F. Allen. Recognizing intentions from natural language utterances. In Michael Brady and Robert C. Berwick, editors, *Computational Models of Discourse*, pages 107–166, The MIT Press, Cambridge, MA, 1983.

[2] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] J. L. Austin. *How to Do Things with Words.* Oxford University Press, New York, 1965.

[4] R. Balzer, L. D. Erman, P. E. London, and C. Williams. HEARSAY-III: a domain-independent framework for expert systems. In *Proceedings of the* $1^{st}$ *National Conference on Artificial Intelligence*, pages 108–110, Stanford, 1980.

[5] Sandra Carberry. Modeling the user's plans and goals. *Computational Linguistics*, Special Issue on User Modeling, 1988.

[6] Sandra Carberry. Plan recognition and its use in understanding dialogue. In Alfred Kobsa and Wolfgang Wahlster, editors, *User Models in Dialog Systems*, Springer Verlag, Berlin–New York, 1988.

[7] David N. Chin. *Intelligent Agents as a Basis for Natural Language Interfaces.* PhD thesis, Computer Science Division, University of California, Berkeley, 1988.

[8] Francisco Corella. Semantic retrieval and levels of abstraction. In Lawrence Kerschberg, editor, *Expert Database Systems*, Benjamin Cummings, New York, 1985.

[9] Daniel D. Corkill, Kevin Q. Gallagher, and Philip M. Johnson. Achieving flexibility, efficiency, and generality in blackboard architectures. In *Proceedings of the* $6^{th}$ *National Conference on Artificial Intelligence*, pages 18–23, 1987.

[10] Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray. GBB: a generic blackboard development system. In *Proceedings of the* $5^{th}$ *National Conference on Artificial Intelligence*, pages 1008–1014, 1986.

[11] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlicka. Unifying data-directed and goal-directed reasoning: an example and experiments. In *Proceedings of the* $1^{st}$ *National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh, 1980.

[12] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the* $5^{th}$ *National Conference on Artificial Intelligence*, pages 58–64, 1986.

[13] J. Robert Ensor and John D. Gabbe. Transactional blackboards. In *Proceedings of the* $9^{th}$ *International Conference on Artificial Intelligence*, pages 340–344, Los Angeles, August 1985.

[14] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The HEARSAY-II speech-understanding system: integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):213–253, June 1980.

[15] Lee D. Erman, Philip E. London, and Stephen F.Fickas. The design and an example use of HEARSAY-III. In *Proceedings of the* $7^{th}$ *International Conference on Artificial Intelligence*, pages 409–415, Vancouver, 1981.

[16] Tim Finin, Aravind K. Joshi, and Bonnie L. Webber. Natural language interactions with artificial experts. *Proceedings of the IEEE*, 921–938, July 1986.

[17] Peter E. Friedland and Yumi Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, 1:161–208, 1985.

[18] Annie Gal. A natural language database interface that provides cooperative answers. In *Proceedings of the Second Conference on Artificial Intelligence Applications*, pages 352–357, 1985.

[19] Alan Garvey, Craig Cornelius, and Barbara Hayes-Roth. Computational costs versus benefits of control reasoning. In *Proceedings of the* $6^{th}$ *National Conference on Artificial Intelligence*, pages 110–115, Seattle, July 1987.

[20] Alan Garvey and Barbara Hayes-Roth. *An Empirical Analysis of Explicit vs. Implicit Control Architectures.* Technical Report KSL-88-21, Knowledge Systems Laboratory, Computer Science Department, Stanford University, March 1988.

[21] Alvin I. Goldman. *A Theory of Human Actions.* Prentice-Hall, Englewood Cliffs, NJ, 1970.

[22] H. Paul Grice. Logic and conversation. In *Syntax and Semantics 3: Speech Acts*, Academic Press, New York, 1975.

[23] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[24] Michael Hewett and Barbara Hayes-Roth. *The BB1 Architecture: A Software Engineering View.* Technical Report KSL-87-10, Knowledge Systems Laboratory, Computer Science Department, Stanford University, February 1987.

[25] Julia B. Hirschberg. *A Theory of Scalar Implicature.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, December 1985.

[26] Aravind K. Joshi. Mutual beliefs in question answering systems. In N. Smith, editor, *Mutual Belief*, Academic Press, New York, 1982.

[27] Aravind K. Joshi, Bonnie L. Webber, and Ralph Weischedel. Living up to expectations: computing expert responses. In *Proceedings of the* $4^{th}$ *National Conference on Artificial Intelligence*, August 1984.

[28] Aravind K. Joshi, Bonnie L. Webber, and Ralph Weischedel. Preventing false inferences. In *Proceedings of the* $10^{th}$ *International Conference on Computational Linguistics*, pages 134–138, July 1984.

[29] Aravind K. Joshi, Bonnie L. Webber, and Ralph Weischedel. Some aspects of default reasoning in interactive discourse. In Ronan G. Reilly, editor, *Communication Failure in Dialogue and Discourse*, pages 213–219, Elsevier Science Publishers, 1987.

[30] M. Vaughan Johnson Jr. and Barbara Hayes-Roth. Integrating diverse reasoning methods in the BB1 blackboard control architecture. In *Proceedings of the* $6^{th}$ *National Conference on Artificial Intelligence*, pages 30–35, 1987.

[31] S. Jerrold Kaplan. Cooperative responses from a portable natural language database query system. In Michael Brady, editor, *Computational Models of Discourse*, The MIT Press, Cambridge, MA, 1982.

[32] Robert Kass. *Acquiring a Model of the User's Beliefs from a Cooperative Advisory Dialogue.* PhD thesis, University of Pennsylvania, 1988.

[33] B. Lewis and E. Goldberg. Emacs transcripts. 1982. Collected in Spring 1982.

[34] Diane J. Litman and James F. Allen. *A Plan Recogition Model for Subdialogues in Conversations.* Technical Report 141, Department of Computer Science, University of Rochester, 1984.

[35] Eric Mays. Failures in natural language systems: application to data base query systems. In *Proceedings of the* $1^{st}$ *National Conference on Artificial Intelligence*, Stanford, August 1980.

[36] Eric Mays. A temporal logic for reasoning about changing data bases in the context of natural language question-answering. In Lawrence Kerschberg, editor, *Expert Database Systems*, Benjamin Cummings, New York, 1985.

[37] Kathleen F. McCoy. *Correcting Object-Related Misconceptions.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, December 1985.

[38] Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, Cambridge, 1985.

[39] Kathleen R. McKeown, Matthew Wish, and Kevin Matthews. Tailoring explanations for the user. In *Proceedings of the* $9^{th}$ *International Conference on Artificial Intelligence*, pages 794–798, Los Angeles, August 1985.

[40] Perry L. Miller. ATTENDING: critiquing a physician's management plan. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(5):449–461, 1983.

[41] Perry L. Miller and Henry R. Black. Medical plan-analysis by computer: critiquing the pharmacologic management of essential hypertension. *Computers and Biomedical Research*, 17:38–54, 1984.

[42] H. Penny Nii. Blackboard systems: blackboard application systems, blackboard systems from a knowledge engineering perspective. *AI Magazine*, 7(3):82–106, 1986.

[43] H. Penny Nii. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53, 1986.

[44] H. Penny Nii. *CAGE and POLIGON: Two Frameworks for Blackboard-based Concurrent Problem Solving.* Technical Report KSL-86-41, Knowledge Systems Laboratory, Computer Science Department, Stanford University, April 1986.

[45] H. Penny Nii, Nelleke Aiello, and James Rice. *Frameworks for Concurrent Problem Solving: A Report on CAGE and POLIGON.* Technical Report KSL-88-02, Knowledge Systems Laboratory, Computer Science Department, Stanford University, March 1988.

[46] H. Penny Nii, Edward A. Feigenbaum, J. J. Anton, and A. J. Rockmore. Signal-to-symbol transformation: HASP/SIAP case study. *AI Magazine*, 3(2):23–35, 1982.

[47] Martha E. Pollack. *Inferring Domain Plans in Question-Answering.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, May 1986.

[48] Martha E. Pollack. A model of plan inference that distinguishes between actor's and observer's beliefs. In *Proceedings of the* $24^{th}$ *Annual Meeting of the Association for Computational Linguistics*, New York, June 1986.

[49] Alex Quilici, Michael Dyer, and Margot Flowers. Detecting and responding to plan-oriented misconceptions. In Alfred Kobsa and Wolfgang Wahlster, editors, *User Models in Dialog Systems*, Springer Verlag, Berlin–New York, 1988.

[50] Robert Rubinoff. Adapting MUMBLE: experience in natural language generation. In *Proceedings of the* $5^{th}$ *National Conference on Artificial Intelligence*, pages 1063–1068, August 1986.

[51] Earl Sacerdoti. *A Structure for Plans and Behavior.* North-Holland, Amsterdam, 1977.

[52] Robert Schulman and Barbara Hayes-Roth. *ExAct: A Module for Explaining Actions.* Technical Report KSL-87-8, Knowledge Systems Laboratory, Computer Science Department, Stanford University, January 1987.

[53] Marc Vilain. The restricted language architecture of a hybrid representation system. In *Proceedings of the* $9^{th}$ *International Conference on Artificial Intelligence*, pages 547–551, Los Angeles, August 1985.

[54] Marc Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the* $5^{th}$ *National Conference on Artificial Intelligence*, pages 377–382, Philadelphia, 1986.

[55] Wolfgang Wahlster, Heinz Marburger, Anthony Jameson, and Stephan Busemann. Over-answering yes-no questions: extended responses in a NL interface to a vision system. In *Proceedings of the* $8^{th}$ *International Conference on Artificial Intelligence*, pages 643–646, Karlsruhe, August 1983.

[56] Bonnie L. Webber. Questions, answers, and responses: a guide for knowledge based systems. In M. Brodie and J. Mylopoulos, editors, *On Knowledge Base Systems*, Springer-Verlag, Amsterdam, 1986.

[57] Robert Wilensky. *Planning and Understanding: A Computational Approach to Human Reasoning.* Addison-Wesley, Reading, MA, 1983.

[58] Robert Wilensky. *Some Problems and Proposals for Knowledge Representation.* Technical Report UCB/CSD 87/351, Computer Science Division, University of California, Berkeley, 1987.

[59] Robert Wilensky, James Mayfield, Anthony Albert, David Chin, Charles Cox, Marc Luria, James Martin, and Dekai Wu. *UC: A Progress Report.* Technical Report UCB/CSD 87/303, Computer Science Division, University of California, Berkeley, July 1986.

[60] William A. Woods. *Semantics and Quantification in Natural Language Question Answering.* Technical Report 3687, Bolt, Beranek and Newman, Inc., 1977.