CONTROL OF MIXED-INITIATIVE DISCOURSE THROUGH META-LOCUTIONARY ACTS: A COMPUTATIONAL MODEL

by

DAVID GRAHAM NOVICK

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

December 1988

APPROVED: [illegible]
Dr. Sarah A. Douglas

An Abstract of the Dissertation of

David Graham Novick for the degree of Doctor of Philosophy

in the Department of Computer and Information Science to be taken December 1988

Title: CONTROL OF MIXED-INITIATIVE DISCOURSE THROUGH META-LOCUTIONARY ACTS: A COMPUTATIONAL MODEL

Approved: [illegible]
Dr. Sarah A. Douglas

Human-computer interaction typically displays single-initiative interaction in which either the computer or the human controls the conversation. The interaction is largely preplanned and depends on well-formed language. In contrast, human-human conversations are characterized by unpredictability, ungrammatical utterances, non-verbal expression, and mixed-initiative control in which the conversants take independent actions. Traditional natural-language systems are largely unable to handle these aspects of "feral" language. Yet human-human interaction is coherent for the participants; the conversants take turns, make interruptions, detect and cure misunderstandings, and resolve ambiguous references. How can these processes of control be modeled formally in a manner sufficient for use in computers?

Non-sentential aspects of conversation such as nods, fragmentary utterances, and correction can be seen reflecting control information for interaction. Such actions by the conversants, based on the context of their interaction, determine the form of the conversation. In this view ungrammaticality, for example, is not a problem but a guide to these "meta" acts. This dissertation develops a theory of "meta-locutionary" acts that explains these control processes. The theory extends speech-act theory to real-world conversational control and encompasses a taxonomy of meta-locutionary acts.

The theory of meta-locutionary acts was refined and validated by a protocol study and computational simulation. In the protocol study, subjects were given a coöperative problem-solving task. The conversants' interaction, both verbal and non-verbal, was transcribed as illocutionary and meta-locutionary acts. The computational model was developed using a rule-based system written in Prolog. The system represents the independent conversational knowledge of both conversants simultaneously, and can simulate their simultaneous action. Simulations of the protocol conversations using the computational model showed that meta-locutionary acts are capable of providing control of mixed-initiative discourse. The model agents can, for example, take and give turns. A single agent can simultaneously take multiple acts of differing control. The simulations also confirmed that conversations need not be strictly planned. Rather, mixed-initiative interaction can be plausibly controlled by contextually determined operators.

This research has application to natural language processing, user interface design and multiple-agent artificial intelligence systems. The theory of meta-locutionary acts will integrate well with existing speech-act-based natural-language systems.

VITA

NAME OF AUTHOR: David Graham Novick

PLACE OF BIRTH: Chicago, Illinois

DATE OF BIRTH: September 3, 1952

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Harvard University

DEGREES AWARDED:

Doctor of Philosophy, 1988, University of Oregon
Master of Science, 1985, University of Oregon
Doctor of Jurisprudence, 1977, Harvard University
Bachelor of Arts, 1974, University of Oregon

AREAS OF SPECIAL INTEREST:

Computer-Human Interaction
Artificial Intelligence

PROFESSIONAL EXPERIENCE:

Research Fellow, Department of Computer and Information Science, University of Oregon, Eugene, 1984-88

Teaching Assistant, Department of Computer and Information Science, University of Oregon, Eugene, 1983-84

AWARDS AND HONORS:

Selected participant, Doctoral Consortium, ACM SIGCHI CHI'88 Conference on Human Factors in Computing

PUBLICATIONS:

Douglas, S. A., Novick, D. G., and Tomlin, R. S. (1987). Consistency and Variation in Spatial Reference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 417-426). Seattle, Washington

Douglas, S. A., Novick, D. G., and Tomlin, R. S. (forthcoming). LingWorlds: An Intelligent Object-Oriented Environment for Second-Language Tutoring. In *Proceedings of the International Workshop on Intelligent Tutoring Systems for Second Language Learning*. Trieste, Italy, 1988.

Fickas, S. F., and Novick, D. G. (1985). Control Knowledge in Expert Systems: Relaxing Restrictive Assumptions. In *Proceedings of the Fifth International Conference on Expert Systems and Their Applications* (pp. 981-994). Avignon, France.

Fickas, S. F., Novick, D. G., and Reesor, R. J. (1985). An Environment for Rule-Based Systems. In *Proceedings of the Conference on Intelligent Machines and Systems*. Oakland University, Rochester, Michigan.

Fickas, S. F, Novick, D. G., Downing, K. L., and Robinson, W. N. (1985). Building Knowledge-Based Systems. In *Proceedings of Northcon/85*, IEEE, Portland, Oregon.

Novick, D. G. (1983). Invasion of Privacy: Development of the Law in Oregon. *Oregon Law Review*.

Novick, D. G. (1984). Computer Software: Legal and Ethical Issues. In *Proceedings of The Computer: Extension of the Human Mind III* (pp. 146-154). Eugene, Oregon.

ACKNOWLEDGMENTS

I wish to express appreciation to Professor Sarah A. Douglas for her guidance, inspiration, perserverance and faith. I thank Professors Scott C. DeLancey, Stephen F. Fickas, and Kent A. Stevens for their constructive, diligent work as members of the dissertation committee. Professor Russell S. Tomlin was very helpful with respect to discourse issues. Barbara J. Grosz graciously reviewed my dissertation proposal and provided many detailed comments; I am grateful for her time and assistance. I received helpful comments from the mentors and members of the ACM SIGCHI CHI'88 Doctoral Consortium. I also wish to thank audiences at Xerox PARC, Information Sciences Institute, and the University of Oregon Cognitive and Decision Science Colloquium for their constructive responses to early presentations of this research. This work was supported in part by a grant from the Foundation for the Improvement of Post-Secondary Education, U.S. Department of Education, number 84.116C, to Professors Douglas and Tomlin, and by a contract from U S West Advanced Technologies to Professor Douglas.

DEDICATION

To the memory of Leona Bachrach Gerard.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This dissertation presents the motivation, methodology, and results of a qualitative study of mixed-initiative conversational interaction. The study examines speech meta-acts which handle the various functions of coördinating and producing conversational discourse. It addresses the problem of multi-agent computer and computer-human interaction by examining human-human interaction. In the day-to-day experience of ordinary human interaction, people encounter and deal with multi-agent communication situations all of the time. If the processes humans use to produce conversation can be understood, these insights may facilitate human-computer interaction that conveniently resembles the interactive discourse of independent human agents.

Human-computer interaction tends to be fragile and frustrating while human-human interaction displays robustness over a wide variety of circumstances (Hayes & Reddy, 1983). Some of the differences between the two forms of interaction are summarized in Figure 1. In human-computer interaction, typically either the human or the computer controls the flow and structure of the interaction. This is *single-initiative* interaction in the sense that only one of the parties is empowered to initiate exchanges. This form of interaction is largely pre-planned, either by the human or by the designers of the program. Even so, it is largely because they are forced by the interface (or have learned from earlier frustration) to use well-formed language that the human and the computer are able to understand each other at all. In contrast, human-human conversations are characterized by unpredictability, ungrammatical utterances, non-verbal expression, and mixed-initiative

	HHI	CHI
control	mixed-initiative	single-initiative
actions	situated	pre-planned
language	informal, relaxed	formal, stilted

FIGURE 1. Differences between interaction modes. Human-human interaction (HHI) is generally more flexible than human-computer interaction (HCI).

control in which the conversants take independent actions. In the face of an unexpected shift of conversational context, somehow humans are able to understand that the shift has occurred. Our every-day language is notoriously ill-formed, full of improper usages, mismatched agreements, halts and starts, in-line correction of self and other, and utterances that may not even be words. Moreover, we interpret and express a wide range of non-verbal behaviors as a concomitant of verbal interaction. In human-human conversation, both conversants have the power to seize the initiative to meet their own needs and expectations. If the language of human-computer interaction seems stilted and tame, the language of ordinary human-human conversation is wild and untamed.

Traditional natural-language systems—even those based on speech acts—are largely unable to handle these aspects of "feral" language because they mainly rely on parsing sentence-level interaction and fail to account properly for the context created by the conversation itself. Conversational processes are apparently not characterized by top-down planning; they are made up of actions which are responsive to the dynamic conversational situation.

A striking aspect of conversational discourse is its coherence—the character of making sense to the conversants. What accounts for the coherent nature of conversation? Despite the incoherence of human-human interaction to an observer to whom only the words are given, the interaction is coherent for the participants; the conversants take turns, make interruptions, detect and cure misunderstandings, and resolve ambiguous references. How can these processes of control be modeled formally in a manner sufficient to bring this sort of coherence to human-computer interaction? Although speech act theory adds clarity to the role of intention in interactive discourse, and implies a central function for intention in making discourse coherent, intention is not sufficient to explain the coherent nature of interactive discourse. I argue in this dissertation that shared models play a key role in the mechanics of conversation and creation of coherence. If interactive discourse is achieved through a process of maintenance of a shared model, the conversants will have to use feedback in their discourse as a process control. This feedback is contained in non-sentential aspects of conversation such as nods, fragmentary utterances, and correction. Such actions by the conversants, based on the context of their interaction, determine the form of the conversation. These actions are thus "meta" to the conversation because they are about the conversation itself rather than constituting its primary substance. In this view ungrammaticality, for example, is not a problem but evidence of these "meta" acts.

This dissertation develops a theory of *meta-locutionary* acts that explains these control processes. The theoretical approach developed and used here is based on a synthesis of two schools of thought with respect to natural language. The first, speech-act theory, is a branch of the philosophy of language.[1] The second, conversational analysis, is a branch of the sociology of language. The differences between these approaches results in what I term the *integration problem.* I propose a solution to the integration problem and then use this result to build an operator-based model of conversation. The theory, which encompasses a taxonomy of meta-locutionary acts, thus extends speech-act theory to real-world conversational control.

The theory of meta-locutionary acts was refined and validated by a protocol study of human-human interaction and a computational simulation of this interaction. In the protocol study, subjects were given a coöperative problem-solving task in a simple domain which necessitated linguistic interaction. The conversants' behaviors, both verbal and nonverbal, were transcribed, and their interaction was mapped into illocutionary and meta-locutionary acts. The simulation was developed using a rule-based system written in Prolog; it embodies a system which, based on the theory of meta-locutionary acts, shows that the theory has practical explanatory power. It extends the work of Power (1979) to meta-locutionary phenomena. Power produced a simulation of two robots who used and recognized illocutionary force in simple conversations. The system for this dissertation contains independent knowledge bases for both conversants and can represent their separate conversational knowledge simultaneously. The conversants simultaneous actions can then be simulated in the rule-based system on a cycle-by-cycle basis. Simulations of illocutionary and meta-locutionary acts in the conversations obtained in the protocol study

[1]Speech-act theory is the source of the terms *locution*, *illocution* and *perlocution.* The spoken words are a locution; illocution is the rhetorical force of the utterance; and perlocution is the change wrought by the illocution. The term *meta-locution* is an invention of the author's which refers to speech about speech. A meta-locutionary act is an illocutionary act which has as its intended perlocutionary effect a change in the the conversation in which the act itself occurs.

showed that meta-locutionary acts are capable of providing contextually appropriate control of mixed-initiative discourse.

The dissertation concludes with an assessment of the computational model, including proposals for extension, and a discussion of the implications of meta-locution for understanding human-computer interaction.

Speech-Act Theory

When language is viewed as a series of assertions about the world, the reasons for speaking and the way language is comprehended are difficult to understand. A breakthrough in understanding the functions of language was made by the development of a theory of speech acts. Austin (1962) noticed that some utterances are more than statements about something; these utterances actually effect changes in the world. For example, the locution "I now pronounce you husband and wife." is not an simply an assertion about the world which can be logically evaluated; rather, under the proper circumstances this statement actually creates the status of marriage between the bride and groom. Austin called this kind of utterance a *performative*. He saw that performatives were acts by speakers, and extended this insight by considering all speech as intention-based attempts to bring about changes in the world. Thus saying "Please pass the salt." is an act of requesting in which the speaker intends that the result of the act will be the listener's passing the salt. This is an appealing approach to multiple-agent interaction because the process of interactive discourse is given motivation and made explicit. Using Austin's terminology, the spoken words are a *locution*; they embody an *illocutionary act* which represents the rhetorical force of the utterance; and the result of the speech act is the *perlocutionary effect*. The theory of speech acts was substantially extended by Searle (1969), who presented a formalized account of the conditions and roles of acts. Viewed as the basis for a computational model of interactive discourse, speech act theory seems to imply a number of requisites for the representation and process. These requirements for conversants include intentions, the ability to reason from intended perlocutionary effects to illocutionary acts, and the ability to respond to illocutionary acts.

Problems of Speech-Act Theory

Speech-act theory has not been without critics. Levinson (1981) argued that speech acts are inadequate for modeling dialogue. First, he argued, the immense variety of surface utterances cannot be reduced to a limited set of acts. Second, speech acts do not appear to correspond neatly to sentences. Third, assignment of meaning to acts cannot be accomplished by a set of rules expressing conventions; rather, this process must involve inferential techniques that take many aspects of context into account. Fourth, sequence-oriented speech-act models are too simple to account for the complexity of real-world exchanges, and coherence cannot, as matter of principle, be explained by rule-based systems. The first three points are addressed directly by the theory advanced in this dissertation. It is plausible that speech acts are specialized by domain. This is analogous to observations of development of sublanguages in communities of speakers with specialized interests or needs. In this case, viewing the process of conversation as a domain itself leads to the observation that speech acts need not correspond to sentence-level discourse units. The circumstance that early speech-act models were simple does not mean that more sophisticated rule-based speech-act models cannot embody situated action in response to complex contexts. As I argue in this dissertation, the fault is not in the idea of speech act theory but rather in confining its application to sentence-level meaning. Finally, Levinson's argument about the computational properties of rule-based systems are mistaken. One could dispute matters such as convenience of representation, but

Levinson's observation that simple rules cannot explain some exchanges does not reasonably lead to the conclusion that this problem is an inherent one. For example, rule-based systems are not inherently limited to modeling adjacency-pair exchanges. Bowers and Churcher (1988), for example, have shown that a rule-based system modeling speech acts can account for complex situated communicative actions. They do so by extending the theory of illocutionary acts from individual performance to social accomplishment; illocution is understood in "dialogical" terms.

Another problem with speech-act theory has been the difficulty with what are called *indirect* speech acts (Perrault & Allen, 1980). The problem which indirect speech acts address is that of explaining why an utterance which on its face appears to be one sort of act and which is nevertheless easily understood as an altogether different sort of act. Indirect acts are often the result of social conventions for seemly behavior and politeness. For example, "Gee, that cake sure looks good." is apparently an act of assertion when the conversants both understand the act as a request: "Please give me a piece of that cake." Understanding and explaining indirect acts is a major open problem for speech-act theory. The protocol analysis in Chapter V makes a modest gain in this area; the evidence suggests that indirect speech acts have utility in terms of efficiency of conversational communication.

There is also a continuing argument over whether conversants must have the ability to recognize illoçutionary acts as such (Bach & Harnish, 1979; Cohen & Levesque, 1986), but the outcome of this debate is not critical to the points made in this dissertation. What is important about speech acts here, though, is a view of language as a series of acts with effects on listeners, or even as coöperative acts which have effects on a jointly constructed conversational structure. Construed broadly, speech acts can be seen as the principal units of of inter-agent interaction; they are the acts which build conversations. Winograd and Flores (1986) describe language as form of human social action. Language as acts, they argue, creates a consensual domain—interlinked patterns of activity. While Austinian speech-act theory was developed to explain overt effects of sentence-level utterances, these requirements can be relaxed to include mental effects in the hearer and sub-sentence level utterances. As I will show in this dissertation, such extensions of speech act theory lead to plausible computational models for interactive discourse.

Extension of Speech-Act Theory

Tying speech acts to sentence level utterances appears to be a limitation which does not take adequate account of the relationship between acts and language. Some acts are physical acts masquerading as speech acts. Others are speech acts which look like physical acts. For example, a light switch which reacts to loud noises might turn itself on when someone yells "Lights on!" but would also work if they yelled "I now pronounce you husband and wife." Conversely, a cough might embody a request for someone to be quiet. In particular, not all effects which speakers intend to cause through their illocutionary acts are physical in nature like passing the salt. As we have seen, some illocutions can make changes in legal relations. Other illocutions have as their effect another speech act. Thus speaking "Do you have the time?" may lead to the hearer saying "Yes, it's ten of two." Indeed, as is centrally important to this dissertation, the perlocutionary effect of an utterance may involve the conversation itself. Thus, for example "Shut up!" may be intended by the speaker to cause the hearer to stop talking. In a more pleasant and coöperative context, speaking "Go on." may be intended by the speaker to cause the hearer to continue speaking (and maybe even effect some purely mental change in the hearer's beliefs about the speaker's level of comprehension). Such utterances can be viewed as meta-locutionary acts because they concern and affect the process of their own conversation. This dissertation, then, develops a framework for research into speech meta-acts which handle the various functions of coördinating and producing discourse as well as research developing semantics and pragmatics for these meta-acts which are sufficiently complete to permit computational modeling of simple interaction in conversational discourse.

Mutuality of Knowledge

The model proposed in this dissertation is based on the idea that the meta-locutionary acts of participants in a conversation are directed toward maintenance of some set of mutual beliefs. Before tackling the problem of what knowledge is shared, though, the question of what constitutes sharing must first be addressed. This issue was studied by Clark and Marshall (1981) in their research on definite reference. They noted what they termed the *mutual knowledge paradox*: People clearly understand definite reference in a finite time, but this ought to take infinite time because to be certain of a reference the conversants are caught in an infinitely recursive evaluation of the form "… B knows that A knows that B knows that A knows that *p*."

Clark and Marshall resolved the mutual knowledge paradox through use of *copresence heuristics*. In the simplest case, *direct copresence* creates mutual knowledge when both parties simultaneously observe a domain feature and are aware that the other party is also observing the feature. For example, two people sitting across from each other at a dinner table observe simultaneously a candle in the center of the table, thus also observing each other observe. Copresence heuristics eliminate the infinite recursion of the mutual knowledge paradox by allowing the parties to have direct knowledge of the conclusion. These direct copresence heuristics are supplemented (and here the supplements may be far larger than the original corpus) by indirect copresence heuristics that provide an equivalent basis for concluding mutual knowledge. That is, direct copresence is not necessary where the circumstances or the parties' general knowledge about each other provide a substitute for physical copresence. Clark and Marshall described an extensive set of *indirect copresences*, the details of which are far beyond the scope or needs of this dissertation. As an example, though, I note the case where two conversants can infer that they are both members of a community (University of Oregon CIS graduate students, perhaps) and from that know that they have mutual knowledge about matters which are generally known to members of the community.

In the case of interactive discourse, the notion of mutual knowledge through copresence heuristics applies directly to conversants' sharing of knowledge about the conversation itself. In conversing, the direct copresence heuristic applies to the utterances of both parties because they hear both their own and the others' utterances while in each other's presence. Thus the conversants have mutual knowledge of their conversation (and they know it, because the mutuality is "on the table"). The conversants also share directly copresent knowledge not only of the individual utterances but also of the contextual structure which the utterances create for the conversation. This, then, is what I mean generally when I refer to a shared model of interactive discourse. Note that this exposition covers only what is meant by sharing and does not reach the issues of what specifically is shared or what is significant in what is shared.

Even though Clark and Marshall's work was based on resolving a problem of understanding definite reference, their concept of mutual knowledge is straightforwardly extensible to other aspects of the domain and the conversants' view of the domain. Moreover, even the nominally limited field of definite reference is broad enough to support reasonable inference about the role of shared structures in interactive discourse. When a conversant makes a reference to a (conversational) domain referent, the effect in the hearer might be a simple one of affording recall of or attention to some specific and insular thing.

More likely, though, is that the referent is one part of a complex memorial structure. If there is a specific and definite referent with copresence, the speaker can reasonably infer the hearer's shared knowledge. But even in such a case the referent is likely to have remindings and shadings of meaning for the hearer which differ from that of the speaker. To the extent that copresence heuristics are available for these remindings (community, shared experiences, "our song") the remindings are part of the shared knowledge of the conversation; beyond this extent, the conversants' knowledge is a matter for inference at best and more likely ignorance. That is, there are often remindings about which the other party has little or no knowledge, and the parties presumably know this.

My point here that the the boundary between shared and unshared knowledge is not sharp. Rather a field of imprecision and conjecture exists where shared dissolves into unshared as the inferences of indirect copresence become increasingly implausible. Thus a shared model of a conversation will be clearest for the facts of the utterances themselves (because of direct copresence) and increasingly indistinct for the memorial extensions of reference and the semantics of the conversants' actions. For another conversant, as for any observer, the imprecision of ascertaining the meanings of a conversant's actions in response to illocutionary acts is exacerbated to the extent that the action is a mental one. That is, some responsive actions are physical—like opening a door or passing the salt; other actions are mental—like remembering Paris or hoping for rain. The speaker who intended to effect a mental action in the hearer has no a priori knowledge of the extent to which an action has occurred.

Given these fundamental concepts of speech acts and mutuality, I can begin to address the research question around which this proposal is centered. A vocabulary, though, does not in itself constitute a theory. In the next chapter, then, I examine the state of related research and the frontier at which my own problem is formed. Specifically, Chapter II looks at the problem of applying speech-act theory to real-world language, examines evidence for shared models of conversations, and discusses the role of repair in maintaining coherence. Chapter III presents the theory of meta-locutionary acts as a way of explaining coherence in terms of conversational acts analogous to domain-level illocutionary acts. Chapter IV discusses the methodology of a protocol study of human-human conversation. In the study, the results of which are reported in Chapter V, a conversational protocol is analyzed in terms of meta-locutionary acts. Chapter VI presents a simulation of the protocol conversation using rule-based operators. The operators represent an application of the theory of meta-locutionary acts; they are able to rely on the theory to account for significant parts of the interaction observed in the protocol. Finally, Chapter VII summarizes the results of the dissertation research, discusses issues of intentionality which arose out the research, and presents thoughts about related future work.

CHAPTER II

SURVEY OF RELATED WORK

The Integration Problem

In addressing the question of what language *is*, speech-act philosophy was spectacularly successful. It solved many problems in linking language to the world and explained why people use language. Speech-act theory has been useful as a basis for some aspects of computational modeling of interactive discourse, particularly in matters relating to intention and planning (see, e.g., Power, 1979; Cohen & Perrault, 1979; Allen & Perrault, 1980; Cohen, 1981). Speech acts could explain in terms of goal-directed behavior how conversants coördinated when and what they said. Speech-act theory reintroduced communication and purpose into analysis of planning in language. It did not, however, prove as successful in explaining *how* language is used. That is, the feedback-suffused protocol evidence of actual speakers was often difficult or impossible to explicate using speech acts (Cohen, 1984; Clark & Wilkes-Gibbs, 1986; Clark & Schaefer, in press). This meant that things like the coördination of real turn-taking could not be modeled with speech acts because the turns frequently did not contain utterances which could be classified as speech acts.

At the same time, socio-linguistic research (see e.g., Schegloff, Jefferson & Sacks, 1977) was uncovering the very linguistic behavior which posed problems for speech-act theory. However, they did not propose computational models. Clark and Wilkes-Gibbs (1986) noted that for both sociologists and speech-act philosophers, the central issue is coördination: How do conversants coördinate on the content and timing of what is meant and understood? The two divergent research traditions had approached the same problems from different directions, equally unsuccessful:

> The issue cannot be resolved by either tradition alone. In the [speech-act] tradition conversation is idealized as a succession of illocutionary acts—assertions, questions, promises—each uttered and understood clearly and completely [citations omitted]. Yet from the [sociological] tradition we know that many utterances remain incomplete and only partly understood until corrected or amplified in further exchanges. How are these two views to be reconciled? (Clark & Wilkes-Gibbs, 1986, p.2)

This problem, which I call the *integration problem*, also bothered Cohen (1984) in his examination of the use of referring expressions in interactive discourse. According to Searle, to perform an illocutionary act an act of predication is required, and the predicate must be uttered. How then, Cohen asked, can one explain the following dialogue where there is no apparent predication for a completed utterance containing only a noun phrase:

> A: Now, the small blue cap we talked about before?
>
> B: Uh-huh.
>
> A: Put that over the hole on the side of that tube .... (Cohen, 1984, p. 104)

The only apparent effect of A's initial utterance can be to direct B's attention to the referent. A does not say anything *about* the referent here. Equally perplexing, Cohen argues, are predications without expressed referents. For example, Sherlock Holmes might lean over a body on the floor and exclaim to Watson "Dead." Conversely, Searle would claim that utterances like "There is a little yellow piece of rubber." contains no act of referring at all and are simply predications. The problem, then, is again that the traditional view of speech acts cannot be reconciled with the linguistic evidence. Cohen noted that the requirement that the act of referring be jointly located with some predication in a sentence or illocutionary act is too restrictive. The functions of reference and predication can be embodied in separate utterances which have, in effect, different though related perlocutionary effects. His solution to this particular part of the problem was to break down Searle's unitary acts into separate acts with separate functions. Thus he suggested that referring, in and of itself, is a kind of request that speakers make of hearer (i.e., to direct attention). But where lies the solution to the general problem of integrating speech acts with the actual character of interactive discourse? The principle of breaking down exchanges involving reference into segments characterized by their goals might be extended to the general phenomenon of interactive discourse by recognizing that each utterance fragment reflects its own proper purposes.

Characteristics of Conversation

The socio-linguistic line of research on interaction is part of a large field which is broadly known as conversational analysis. This research has described a wide range of conversational characteristics which are not directly representable in Austinian speech-act theory. These characteristic behaviors include laughter, fragmentation, turn-taking, correction, gaze, and nonverbal communication generally.

The prevalence in conversation of laughter and fragmentation was shown by Allen and Guy (1974). There are distinct regularities associated with these phenomena. For example, in conversations between a male and a female, the male tends to laugh twice as often as the female. Although fascinating, laughter is really beyond the scope of the present research. Fragmentation, however, is one of the central characteristics of language for which this dissertation attempts to account. This includes uncompleted sentences and words, plus non-word utterances such as "uh," "ah," and "eh." Fragmentation is sometimes associated with correction (Allen & Guy, 1974). More generally, they represent sub-lexical, incomplete sentential thoughts, or complete conversational contributions which are expressed in non-sentential form. These include, for example, referential and confirmatory utterances such as "OK, now, the small blue cap we talked about before?" (Cohen, 1984, p. 104) and "Uh-huh." That is, while fragmentation sometimes results from correction, it is a general functional characteristic of natural discourse. Thus despite their part-formedness, fragments are used in utterances which are nevertheless understood by conversants.

Turn-taking (Sacks, Schegloff, & Jefferson, 1974; Duncan, 1980) is a control-oriented account of conversation that explicates patterns of mixed-initiative interaction in terms of conversational turns. This represents an organizational substrate for domain-level, intentionality-based interaction. It is maintained through a wide set of behaviors and acts, including nonverbal communication (Ekman & Friesen, 1981).

In conversation, repair of language and interaction has been observed in two forms: self- and other-correction. There is an apparent preference for self-correction (Schegloff et al., 1977; Clark & Wilkes-Gibbs, 1986), which leads to fragmentation of utterances as speakers in effect erase parts of their utterances and replace them with substitute phrases. As a sub-sentential phenomenon, self-correction is not accounted for by speech-act theory. Other-correction, when lexical, for example, presents the same difficulty.

The role of gaze in conversation has been the subject of extensive—though largely descriptive—analysis. Gaze performs multiple functions in human-human interaction (Argyle & Cook, 1976; Argyle, Ingham, Alkema, & McCallin, 1981), and can be described in terms of its temporal association with language use in dialogue (Beattie, 1981). Gaze was found to be organized in a coördinated system with the plans underlying speech and with the speech flow. Gaze was more highly associated with turn-yielding cues than simply with syntactic clause boundaries.

Gaze is a significant part of a broader range of behaviors generally considered as nonverbal communication. Extensive systems have been developed for noting physical states and actions associated with language (See e.g., Birdwhistell, 1970; De Long, 1983). Indeed, there is much evidence in support of the proposition that nonverbal behaviors are *part* of language. For example, Hoffer and Santos (1983) showed that patterns in and meanings of nonverbal communication differed by culture and dialect. For American English, the verbal and nonverbal channels cannot be unlinked:

> All of the emerging data to me to support the contention that linguistics and kinesics are infracommunicational systems. Only in their interrelationship with each other and with comparable systems from other sensory modalities are the emergent communication systems achieved. (Birdwhistell, 1970, p. 127)

Clearly, it would be difficult to explicate coherence in interactive discourse without taking account of these characteristics of language as it is used; this is the heart of the integration problem. Why, though, do conversants rely on these functions? In the next two sections, I address this question by looking at models of mixed-initiative conversation which would explain the use and necessity of these behaviors.

Evidence for Shared Models of Conversations

This section examines the role of feedback as a process control for interactive discourse by discussing the following questions: How closely does the hearer track the speaker's utterances against their shared assumptions about the conversation? How do conversants minimize and/or correct expectation failures?

The evidence is strong for the proposition that conversants in interactive discourse share a model of their conversation. In informal terms, a weak version of this conjecture would be that the conversants must have at least some knowledge necessary to the conversation which is common to all conversants. A strong version is that the conversants are jointly creating a single (though possibly complex) intellectual product. Suchman (1987) characterizes conversation as an "ensemble" work:

> Closer analyses of face-to-face communication indicate that conversation is not so much an alternating series of actions and reactions between individuals as it is a joint action accomplished through the participants' continuous engagement in speaking and listening [references omitted]. (Suchman, 1987, p. 71)

Clearly, speakers of a language necessarily share lexical knowledge, even if their internal representations of the lexicon differ in their extension. To what extent, then, must conversants construct or share a mutual model of their jointly-created conversation?

Implications of the Interactive Process for Shared Models

Observation of interactive discourse suggests that when conversants share information, their exchanges cover the range of conversational levels, from domain information to turn-taking. As has often been noted, one of the central questions of interactive discourse is how the conversants coördinate the timing of what is meant and understood (see e.g., Clark & Wilkes-Gibbs, 1986). Clearly, a conversational model common to the conversants would facilitate this coördination. This process has been characterized as one of coöperation. The fact of coöperation would seem to presuppose something to coöperate about. According to Clark and Wilkes-Gibbs (1986), Grice (1975) observed that conversants coöperate in their contributions to a conversation by directing their contributions toward the *accepted* purpose or direction of their exchange.

Even in the simplest cases, though, it is apparent that some kind of mutually understood model of the conversation is created. Consider the railway station protocols discussed by Allen and Perrault (1980):

> Patron: The 3:15 train to Windsor?
>
> Clerk: Gate 10. (Allen & Perrault, 1980, p. 442)

Using the notion of mutual knowledge developed by Clark and Marshall, the clerk and the patron mutually know that the patron has made an inquiry about the 3:15 train to Windsor. This means that the clerk can rely on the patron's reasonable acceptance of a coöperative response by the clerk. Conversely, assuming that the clerk is being coöperative, the patron can expect that the clerk's next utterance will be responsive to the query the patron made. Thus even in this terse exchange the conversants can be viewed as relying on a mutually known model of the structure and direction of their conversation in the sense that they use indirectly copresent community knowledge about exchanges between patrons and clerks.

Cohen and Perrault (1979) also described a model for possible intentions underlying speech acts. Cohen and Perrault were interested in providing a theory of speech acts which, among other things, answered questions about changes the successful performance of a speech act makes in the speaker's model of the hearer and in the hearer's model of the speaker. They proposed a planning approach which provided formal criteria for defining speech acts in terms of intentions, abilities, and effects. Acts of requesting and informing were then specified as operators which can be used by a planning system. This model necessarily assumed a speaker's model of hearer and vice versa. To the extent that these are self-referential we have at least the raw basis for a shared model, whether or not the authors recognized it as such.

To be coherent, a conversation must be about something. It follows that the discourse segments which make up a conversation must have corresponding foci. The notion of focus as a characteristic of interactive discourse was developed by Grosz (1977). These studies were primarily based on task-oriented dialogues. In the context of shared models, a practical interpretation of Grosz's work is that focus really means the thing on which the conversants are mutually focusing; otherwise the conversation would lose its coherence (absent recurring and unbelievable coïncidence). Grosz's idea of focus space implies that this is a shared view or partitioning of the domain; indeed, discourse *presumes* mutual focus. Another way of looking at focus spaces might be to consider them as shared models of the real domain which provide necessary (1) common extensional meaning and

(2) reference points for coherent discourse structure. Similarly, in the protocols studied by Clark and Wilkes-Gibbs (1986), the conversants (have to) find a mutually acceptable perspective. As I see this result, the parties have established an interpretive model of the real domain that constitutes a suitable model of the domain for their achieving their underlying intentions through discourse.

Of course, the role of focus and perspective need not be limited to physical domain objects, definite referents of any kind, or even pragmatics. It has been noted that a successful speech act is based on agreement or on continuing negotiation of what values should prevail (Jernudd & Thuan, 1983). That is, the conversants must arrange to share the meanings of their illocutionary acts, even if the process of understanding doesn't require that the acts be recognized explicitly. In order to maintain coherence, then, conversants have (or are trying to obtain) a shared model of at least the semantics of the illocutionary acts. This sort of negotiation may also be applicable (and probably much more frequently) to the pragmatics of a conversation.

The Role of Meta-Locution in Conversation

Participants in interactive discourse presumably engage in their interaction because of underlying intentions. While they may have private intentions which motivate their communication, they present to each other apparent discourse purposes (Grosz & Sidner, 1986). Thus if the discourse purposes are not apparent, their meanings must be clarified. This secondary discourse is a collaborative process which is meta to the subject of the conversation. Similarly, negotiation of referential meanings becomes meta to the conversation. For the meanings of illocutionary acts, the evidence of negotiation is indirect. Jernudd and Thuan (1983) suggested that speech acts, to be successful in language, require agreement by the conversants on the meaning of the acts. They observed that partners in communication generally coöperate in the (meta-) communicative goal that the speaker's speech act is identical to that understood by the hearer. In other words, if both conversants are striving to make sense of their conversation, they will try to get the speaker's and hearer's identification of the speaker's act to match up.

In the same way that the semantics of illocutionary acts are subject to negotiation and agreement, so too is reference a collaborative process. This was shown experimentally by Clark and Wilkes-Gibbs (1986). The conversants must mutually accept that the hearer understands the reference before the conversation proceeds. As I understand this work relative to possible shared models of the conversation itself, there is an incremental process of building a mutually understood reference scheme that constitutes a shared model at least as to extensional meanings.

Consistent with an interpretation of Clark and Wilkes-Gibbs's work showing the negotiation of the topic of a conversation (and thus suggesting a shared conversational model) is Grosz's view that the knowledge of the participants in discourse can be characterized by a common structure. She suggested that one can model focus in discourse as a partitioning of a semantic net which encodes the domain of discourse (Grosz, 1977). This network represents one conversant's model of the conversation, and it therefore includes that part of the conversation which the conversant believes constitutes the shared conversational structure. Of course, the conversants' respective models need not be and are not likely to be identical. This is like Clark and Wilkes-Gibbs's notion that hearers aren't trying to assure perfect understanding of every utterance; they just need understanding that is adequate under the circumstances. This was recognized by Grosz in trying to distinguish knowledge from belief. It should be noted that Grosz presents what I would call an extremely integrative view of the shared model. This view is an especially appealing one in the context of Clark and Marshall's notions of mutual knowledge because it incorporates naturally the consequences of the application of direct copresence heuristics to the conversation itself. Grosz's (1981) analysis indicated that in their speech conversants *assume* a common focus; they usually do not have distinct models of each other's focus. This has the salutary effect of avoiding the infinite recursion of the mutual knowledge paradox. She suggested that the speaker assumes that the hearer, in understanding an utterance, has followed any shift in focus.

Grosz's analysis looked at explicit indicators of shifts in focus; a similar analysis could be applied to other utterances or acts which have specific model-maintenance functions, and even to functions which conversationally propose lack of mutuality. Such model-maintaining utterances are what I call have called meta-locutions, which correspondingly embody *meta-illocutionary acts*. In other words, the intended perlocutionary effects of such acts concern the process of conversation itself rather than the underlying discourse purposes. The utterances are in this sense meta-locutions because they are about the process of the very conversation in which they occur. For example, an utterance like "Go on ...." can be seen as a meta-locutionary act in which the illocutionary meaning is something like "I'm asserting that I don't want to repair anything here and I'm letting you keep your turn." Similarly, looking away while talking could be construed as a meta-locutionary act such as "I'm holding on to my turn." Such functions of gaze in particular have been recognized as providing meta-information used in conversational control. People link use of their verbal-auditory and nonverbal-visual channels:

> (1) Since the vocal channel requires that people take turns to speak, signals must be used to negotiate turn-taking; it would be difficult to contain these signals in the vocal channel—speakers would have to combine messages and meta-messages, and listeners would have to speak at the same time. Therefore the synchronizing signals are forced into the second channel. (2) While a person is speaking he needs feedback on how others are reacting; this could be provided by vocal comments, but that would involve double-talking, so feedback signals are also relegated to the second channel. (Argyle & Cook, 1976, p. 124)

The relatively broad range of behaviors which constitute communicative interaction, then, suggest a pervasive and central role for meta-locution in the control of conversation.

Interactive Discourse Requires Monitoring by All Conversants

Jernudd and Thuan (1983, p. 81) reasonably contended that language is usually expectation driven: "Norms of use are founded on expectations that users form. Obviously, interaction proceeds mainly in worn grooves and these generate reasonable expectations." Applying this principle to the maintenance of shared conversational models, this suggests that conversants often share a large part of their conversational model automatically or by default. A consequence of this is that if a conversation is to be coherent each conversant must have a set of expectations which is consistent with the others'. These sorts of structures have been recognized when conventionalized as analogous to the well-known script models of conversations (Cohen, 1984). Yet not all, and maybe not even most, conversations always follow the script exactly. Otherwise we would find, contrary to experience, that we are always having the same conversations or fragments over and over again. This means that along with the shared set of expectations, conversants must detect and maintain the set of deviations from the well-worn expectational grooves. Thus Clark and Wilkes-Gibbs (1986) noted the prevalence of conversational feedback: the hearer lets the speaker know how things are going. This implies that the hearer has a model of what the speaker is *trying* to say. This process of monitoring thus apparently involves the hearer checking the actual utterances of the speaker for consistency against a set of expectations. The deviations from the expectations are, as I've observed, frequent. Moreover, no small set of expectations could possibly cover the variety of conversations in which we might and do engage. As a consequence the identification and selection of expectations also becomes important.

Repair-Based Conversational Interaction

We thus see that for conversation to be coherent in a manner consistent with the observed process of conversational monitoring, the conversants must maintain adequate models of the discourse. To the extent that the conversants are having the *same* conversation (i.e., to the extent that the conversation is coherent), the model must be a shared one. This does not mean that the models must be identical. Rather, if conversants satisfice with respect to understanding, their conversation can be, for each of them, coherent to the extent that their models are believed to overlap. For example, the sort of conversation in which one person's down-to-earth discussion is interpreted by the other conversant as a metaphor or parable is coherent for both participants, even though their models may share only an analogical structure. If a conversant detects too great a divergence between her model and the apparent track of the conversation, she may take remedial action to regain mutuality of conversational knowledge.

As previously noted, Grosz (1981) showed that in interactive discourse conversants have a pervasive assumption that they share a common focus. This approach in effect substitutes inference for actual mutual knowledge of focus. I note, though, that the word "assumption" may be distracting. The assumption of mutual focus is usually true. This phenomenon is associated with the highly predictive nature of discourse. The assumption fails only when the expectations are not met (and thus the focus turns out not to be mutual). How often does this occur? How do conversants minimize and/or correct these failures? Moreover, the mutual-knowledge assumption applies not only to focus but to many (if not all) aspects of interactive discourse. Grosz specifically demonstrated the existence of the assumption for focus, but the factors which make the assumption occur with respect to focus are also present for most of the aspects of a shared model of conversation. Grosz observed that the speaker is always one step ahead of the hearer (simply because the speaker is speaking), and noted that communication only ensues if shifts in focus are in fact clearly indicated to the hearer. It is true that listeners' predictions of what speakers say is sometimes (or even often) correct; however, listeners cannot confirm their understanding of their conversational model as mutual until their prediction has been realized. Grosz suggested that the main avenue for understanding this process is through mechanisms that distinguish the conversants' beliefs and then reasoning about knowledge and beliefs. But generalizing the shared focus process to the shared model process, if the speaker is one step ahead of the hearer then how big are these steps? Keeping the steps small minimizes the size of the failures of expectation. This is turn keeps the mutual knowledge assumption true enough to obviate the need for an elaborate maintenance scheme.

Correction, Repair, and Feedback Generally, are Pervasive Phenomena of Interactive Discourse

In one sense, all interactive discourse is feedback. That is, the utterances of one conversant are recursively responsive to the utterances of the other (see e.g., Hobbs, 1985). This view, though, is too broad to be useful. Much of this sort of recursion relates to high-level, domain-based information. It is the purview of traditional Austinian speech act analysis. In contrast, I am interested here in the kind of routine feedback which is below the domain-knowledge level—the kind of feedback which concerns the process of the interaction itself. For example, speakers have been shown to adjust their production of language to account for the non-verbal actions of their hearers (Hopson-Hasham, 1983). One might view conversation as an inter-dependent set of simultaneous conversations, with each sub-conversation pertaining to a different level of interaction. Thus at the "top" level—that of traditional speech acts—we see the body of conversation which is the focus of conscious interaction, the *point* of the exchange. At lower levels we find conversations about the levels above until, at some "bottom" level, there isn't anything left to discuss. Figure 2 depicts such a set of levels, whose attributes reflect identifications in the discourse and socio-linguistic literature of skills or domains pertinent to mixed-initiative conversational interaction. The levels are not hierarchical; rather, they are intended to represent a set of simultaneous and mutually facilitating processes. Thus at the top level, conversants interact in the realm of their underlying goals—the domain is the set of their possible actions in the world and overall conversational intentions. As part of their interaction, the conversants will need, following Clark and Wilkes-Gibbs (1986), to resolve references to things in the top-level domain, hence the inclusion in the figure of a separate level for acts which accomplish this. Next, following Jernudd and Thuan's (1983) notion of dynamic interpretation of illocutionary acts, there is a level for meta-locutionary acts to resolve these meanings. The ellipsis in the figure stands for a broad range of levels corresponding to the sub-domains of various conversational skills which are accomplished though actions which affect the conversation. Finally—in the figure, although not necessarily primitive in the real conversational process—there is a level for the turn-taking acts reported by Sacks et al. (1974). The "lower" levels permit conversants to accomplish the more complex tasks represented by "higher" levels by maintaining coherence in the interaction. In its analysis of the processes of the control of language, this dissertation, then, looks at feedback from the bottom up. The usual problems of reasoning about domains are not presented.

To illustrate the woven nature of these layers of interactive discourse, here is a brief excerpt from a protocol of an English-as-a-second-language lesson. The layers and the analysis are set out here for observational purposes rather than as a specific theory of linguistic interaction. This protocol is interesting because the context necessitates that the

Conversation effectuating domain intentions
Perlocutionary effect: domain-level actions and changes in belief structures
Illocution: Austinian speech acts
Locution: sentence-level utterances

Conversation effectuating domain reference
Perlocutionary effect: changes in extensional reference
Illocution: attention-directing speech acts
Locution: interjected phrases, deixis

Conversation resolving illocutionary meanings
Perlocutionary effect: agreement on meanings of language acts
Illocution: indications of understanding or misunderstanding
Locution: corroborative restatement, repetition

. . .

Conversation managing turn-taking
Perlocutionary effect: agreement on who should be talking
Illocution: interruption or indication of super-level agreement
Locution: start and stop signals such as directed gaze, gesture, nodding

FIGURE 2. Possible levels of conversational interaction. Each level represents interaction which maintains models of the levels above.

conversants arrive at new agreements about the labels and meanings and roles of various things, including illocutionary acts. The English-speaking teacher (T) and the non-English-speaking student (S) sit on opposite sides of a small table. Various cardboard tiles, depicting geometric shapes which are large or small, blue or red, circular or square, lie at one edge of the table.[2] The teacher and the student engage in the following discourse (non-verbal actions are described in brackets and emphasis is indicated by underlining):

(1) T: [Puts cards LBC LRC SRS in the center of the table.]

(2) T: First can you show me [Makes 'pointing' gestures.] the circle.

(3) T: Which one—

(4) T: [Glances down and up.] I'm sorry the square—

(5) T: which one is the square.

(6) S: [Looks confused. Looks at T, arms at side.]

(7) S: Square.

(8) T: Uh huh.

(9) S: [Points to LRC.]

(10) T: The square.

(11) S: [Points to SRS.]

(12) T: OK, that's right.

(13) T: That's a square. [Pointing to SRS.]

(14) T: This is a circle. [Pointing to LRC.]

(15) T: These are the circles. [Pointing to LBC and LRC.] (Novick, 1986, p. 1)

This exchange exhibits a number of interesting features which can present the reader a more concrete idea of the general role of layers of discourse and the phenomena they represent. These features include the rapid establishment of the meaning of "show" in (2); T's self-correction in (4); T's confirming repetition in (5); S's initial failure to take his turn in (6); S's indication of non-comprehension by repetition in (7); S's purely deictic language

[2]In the transcript, the notation refers to L(arge) or S(mall), B(lue) or R(ed), C(ircle) or S(quare) shapes. Thus the LBC is the large blue circle, the LRC is the large red square, and the SRS is the small red square.

acts in (9) and (11); T's indication of failure by repetition in (10); and T's holding on to her turn in (13), (14), and (15). We know that in this exchange a person is teaching English to a non-English speaker. Thus (2) encompasses both locutions and meta-locutions, and (6) is clearly some sort of communicative act but must be meta-locutionary. Interestingly, in the absence of deixis, none of these utterances can be considered standard Austinian speech acts. That is, contextually determined or physically indicated references stand in for the explicit references which would be needed for Austinian analysis. Rather, T and S demonstrate a kind of mutual control of their discourse through a heterogeneous mixture of acts, most of which appear to track the conversants' comprehension and acceptance of previous acts.

Even though it turns out that we engage in this sort of feedback-saturated conversational behavior every day, rarely are we conscious of it. Schegloff, Jefferson, and Sacks (1977) pointed out that, although the phenomenon has largely been ignored by linguists, the phenomena of correction and repair occur massively in conversation, the most common use of language. Feedback is more than a surface effect, and more than a window into the mechanisms of natural language understanding and generation (albeit a valuable one through which I propose to peer). Feedback is, in fact, part of the mechanism or process of conversation in itself. Although some research has looked at positive feedback (see citations in Clark & Marshall, 1986), most studies of feedback in interactive discourse have focused on issues of correction and repair. Thus while the comments of researchers in this area are largely phrased in terms of correction and repair, I think it is fair to extend the thrust of their conclusions to feedback generally . One reason for such extension is the recognition that positive feedback is likely to be simply a conversant's assurance that repair isn't needed. An emerging sociological view of language recognizes feedback as an important part of the conversational process. Jernudd and Thuan (1983) suggested that correction, conceived of in the broadest sense, is built into the core of language and is integral to the structure of all communicative acts. Speakers' resources include the ability to make use of such built-in correction devices; a correction system that permits them to repair utterances; and variable access to a more rigorous societal system of correction in the form of language treatment and language planning. Conversants cannot anticipate every possible miscommunication in advance of its occurrence. Rather, they work moment-by-moment to identify and remedy the troubles which inevitably arise in mixed-initiative discourse (Suchman, 1987). In other words, when conversants use the broad set of tools available to them, they use tools which are built around the notion of feedback.

Why should feedback play such an important part in the process of interactive discourse? While some aspects of discourse are settled before a conversation begins, many others remain to be determined as part of the interaction itself. Aspects of discourse that are usually not the subject of correction or repair may nevertheless involve feedback, either through positive feedback indicating acceptance of normative values or in the exceptional case through repair. Thus conversants normally consider parts of discourse like the lexicon and the set of speech acts to be relatively fixed. They can indicate agreement (or at least not indicate disagreement) as long as they do not encounter new words or acts, or as long as previously encountered words or acts are used with their conventional meanings. Thus a successful speech act is based on agreement—the normal case—or on ongoing negotiation of what values shall prevail (Jernudd & Thuan, 1983). In other words, the conversants' valuations, beliefs, and purposes may converge or conflict with each other's. To the extent they converge, the discourse will manifest agreement; to the extent they diverge or are unclear, the discourse will manifest negotiation. Where, after all, do the meanings of things like speech acts come from? Conversants need to find out each other's expectations of the meanings of speech acts, need to express their own such expectations, and need to find a way to agree on these. The extent to which a speaker is successful in producing a speech act depends on the extent to which the conversants agree it shall be so. This agreement depends on shared expectations of speaking and language. The conversants' understanding of the speech act reflects the (historical) resolution of negotiation of fairly permanent expectations (Jernudd & Thuan, 1983).

Some aspects of discourse are of course not susceptible of normative predetermination. One such aspect is reference. It turns out, even in situations where both conversants can perceive the referents used in their discourse, that definite reference in interactive discourse is a collaborative process requiring actions by both speakers and hearers (Clark & Wilkes-Gibbs, 1986). Another aspect requiring feedback is the turn-taking behavior characteristic of interactive discourse described by Schegloff et al. (1977).

Other aspects of interactive discourse requiring feedback include most if not all of the structural qualities of discourse. The speaker's generation process may even include a sort of self-feedback or "monitoring" as he listens to himself talk. With respect to repair-oriented feedback, Jernudd and Thuan (1983) pointed out that kinds of feedback from hearers to speakers include production errors that escape the speaker's monitor, nonreceipt of what was said, incomprehension, miscomprehension, disapproval, and perhaps more. With respect to positive feedback, Clark and Wilkes-Gibbs (1986), along with many others, observed that sociologists have shown that when one person speaks, the others not only listen but let the speaker know they are understanding—with head nods, "yes's," "uh-huh's," and other so-called back-channel responses.

The role of feedback seems to be linked directly to process of language generation. Sociologists of language have observed that speakers have to have a repertoire of ways of following their own generational processes. This repertoire will involve speakers' abilities to monitor, correct, evaluate, and correct what they are producing even as the process takes place. They need a way of checking that what they are actually saying is consistent with what they intend to say. Additionally, they need to cope with the reactions of the hearers (Jernudd & Thuan, 1983). A large class of nonverbal behaviors is used by conversants for such feedback. Ekman and Friesen (1981) described a class of nonverbal behaviors which they termed *regulators*:

> These are acts which maintain the back-and-forth nature of speaking and listening between two or more interactants. They tell the speaker to continue, repeat, elaborate, hurry up, become more interesting, less salacious, give the other a chance to talk, *etc*.... The most common regulator is the head nod, the equivalent of the verbal mm-hmm; other regulators include eye contacts, slight movements forward, small postural shifts, eyebrow raises, and a whole host of other nonverbal acts. (Ekman & Friesen, 1981, p. 90)

These behaviors convey feedback so intrinsic to interaction that conversation stops if one of the conversants suppresses them (Ekman & Friesen, 1981).

Schegloff et al. (1977) also pointed out that because of the overwhelming evidence for correction and repair in conversation, any adequate theory of the organization of natural language will have to account for how natural language handles its intrinsic troubles, including the organization of repair. In this view, repair (specifically, and, I suggest by extension, feedback generally) is an inherent part of the process of interactive language. Douglas (in press) examined protocols of interactive communication between tutors and students; she suggested that the protocol data indicate that conversational breakdown and consequent repairs may affect the selection and sequence of domain-level content. This interaction displays a local organization that is dynamic—not pre-planned—and yet through the repair process generates global coherence.

"Non-grammaticality" and apparent errors in discourse are thus not to be explained or erased by grammars of non-grammaticality that derive spoken language from a perfect formulation[3]. These characteristics of interaction are the result of performance and cannot be accounted for by extension of competence-based sentential grammars. Rather, these "imperfections" are phenomena to be explained in and of themselves, and are thus useful objects of study in the search for scientific understanding of language.

Intention, Action, and Language

An enormous amount of work in natural language processing, and in artificial intelligence generally, assumes the existence and utility of human intentionality. This work suggests, more or less explicitly, that actions in the world are the result of humans' intentions. Speech-act theory itself is based on this sort of assumption because illocutionary acts are produced by speakers to achieve intended perlocutionary effects: use of language is a form of intentional action (Searle, 1969). Nevertheless, the relationship between intention, action and language is not well understood. For the research presented in this dissertation, two issues are particularly problematic: First, what (linguistic) behaviors are intentional? Second, how do people act on intentions to produce conversational interaction? I address each of these problems in turn.

Acts and Signals

Aside from the occasional case like someone crying out in surprise or fright, verbal acts are largely considered to be intentional. At the same time, there is a class of behaviors, including communicative behaviors, which are widely regarded as unintentional or unconscious. As I have discussed, there is a wide range of nonverbal behaviors in conversational interaction. These behaviors can be interpreted either as intentional acts or

[3]Cohen (1984) presents a brief account of grammar-based approaches to ill-formed input.

as unintentional signals. To some analysts, the majority of this massive stream of communication is unconscious on the part of the agent (Allen & Guy, 1974). To others, significant aspects of nonverbal behavior are directly intentional (Argyle & Cook, 1976; Birdwhistell, 1970). It is certainly true that the kinds of routinized behavior relevant to conversational control are in an indistinct zone with respect to intentional action. They seem to be on the periphery of awareness (Ekman & Friesen, 1981). Some communicative behaviors seem to be in the province of autonomic response; pupil dilation and contraction have been observed in response to informational content (Argyle & Cook, 1976). Nonverbal behaviors are also interpreted as involuntary because they convey or reveal things which the agent has no intention of communicating (Allen & Guy, 1974). Yet other behaviors seem to be part of the same action that we associate with production of an utterance; the rise or drop in pitch at the end of English sentences is invariably accompanied by a raising or lowering of the eyelids, head, or hands (Scheflen, 1980).

The issue is further complicated by the possibility that conversants can consciously display behaviors that would ordinarily be unconscious. This can be done for emphasis (e.g., looking up in frustration) or as a deception (e.g., looking blank to feign lack of prior knowledge of a reference).

From the standpoint of understanding, nonverbal communication the situation is equally perplexing. Actions which could be taken as cues are not always noted by the partner (Allen & Guy, 1974). For example, the extent of people's attention or perception of gaze appears to vary widely (Argyle & Cook, 1976). In short, the role of intention in producing nonverbal communicative acts is an unsettled matter:

> What is actually intentional and what is not need not by any means be the same as the way it is treated by others. Accordingly, although it is fruitless to try to decide what messages a person *actually* intends to convey and what he does not, how people treat each other in this regard should neverthless be carefully attended to. That is, it is very important to consider what aspects of of the flow of

information participants treat *as if* they have been provided intentionally and what aspects they treat *as if* they are unintentional. As a corollary to this, it then becomes a matter of great interest to investigate which features actions must have to be treated as intentional and which they must have to be treated otherwise. To the best of my knowledge, this question remains one to be investigated systematically. (Kendon, 1981, p. 10).

There is no getting around the fact that intentionality is a difficult subject. How, then, is intention to be interpreted in a computational model of conversation? The characterization of much communicative behavior as unconscious is the product of introspective analysis in which the analyst cannot locate any specific intent or purpose for motor activity. In my view, this conclusion is the product of an ill-founded assumption that unconscious equals unintentional. In defining intentionality, for example, Ekman and Friesen (1981) specifically refer to the "deliberate" use of a nonverbal act to communicate a message to another informant, although they do note that it may not be possible to determine the intentionality of every instance of nonverbal behavior. To shed some modest light on this matter, I obtained from a variety of adult informants descriptions of their own processes of linguistic production. All felt that they had little or no conscious control over the process of actually producing speech; they could not explain how they talked. This experience, I feel, is the product of the routinized nature of linguistic production; people spend a great deal of time using language. Yet none of the informants would characterize their speech as involuntary. Even if they were unable to articulate their intentions, they surely had motivations for speaking, even on an utterance-by-utterance basis. There is no reason to distinguish the nonverbal acts associated with these utterances as any less the product of such motivations. It is not necessary to specify goals for the acts in order to describe them (Duncan, 1980). But to understand their part in the process of interaction, one needs to connect them to the motivational structure which accounts for the utterances which they accompany or affect.

There is perhaps a reasonable analogy here between the production of language and the performance of other motor activity. If I walk from my desk to the bookshelf to get a book, I am performing an action in service of my intention to get the book. The overall intention may or may not be conscious, but certainly the individual actions which accomplish it—using my legs and feet, maintaining my balance—are not consciously performed. But neither are these actions involuntary; they are simply easy and routine. It is possible that I might move my leg reflexively if, for example, someone spilled ice-water on it. Similarly, I might blink if dust irritated my eyes. But in a purposive context, both moving my leg and blinking help me to achieve non-reflexive goals. They are the consequences of intention and constitute its embodiment. That is, while the act of walking to bookshelf can be said to embody my intention to get the book, this "act" is a composite; it has no existence outside our interpretation of the sum of a large number of smaller acts which together produce it. Even if attenuated, intentionality must underlie each constituent sub-act. Thus while the distinction between voluntary and involuntary linguistic action is not clear, a reasonable model of conversation will interpret displayed behaviors in terms of intentional acts unless (1) they can be shown reflexive because of physical factors or (2) they do not occur in—or appear to be reasonably related to—a context of larger, intentional action.

Planned vs. Situated Action

How do people's intentions produce conversational acts? More specifically, how does intention get translated into a sequence of acts that produce coherence in the organization of conversation? A strong thread in artificial intelligence has involved production of rationally organized actions through planning. Planning systems have been proposed for the production of text (McKeown, 1985) and for interactive conversation (Power, 1979;

Hobbs & Evans, 1980; cf., Johnson & Robertson, 1981).[4] Other computational models of interactive discourse suggest that conversants use similar processes which rely on depth-first tree search. Grosz (1981, 1982) implied this by using a stack-based process for changing focus spaces in conversation. It should be noted, though, that Grosz does not at all claim that conversation is pre-planned. The structure of a discourse, she observed, tends to arise naturally out of the structure of the discourse task. It is the focus-space structure which models the conversation as it develops that is stack-based. Reichman (1985), though, directly proposed an ATN-based model for conversational exchanges.

It is unlikely that simple planning or stack-based models of interactive discourse are adequate for modeling mixed-initiative conversational discourse. The importance of conversants being able to change the structure of their conversation in unforeseen, flexible ways is underlined by the observation that although some tasks produce neatly stacked discourse structures, everyday interaction requires an enormous variety of structures for which stack-like models are inadequate. The principal problem is that planning involves searching a state space, yet for most conversation the future states of the conversation are not reasonably calculable. For example, Birnbaum (1986) pointed out the case where a conversant refutes an argument on the grounds that the other conversant has used a supporting fact which is demonstrably false. To have planned this exchange from the beginning, the conversant's original state space would have had to include not only the universe of all relevant facts which support her opponent's argument but also all other possible facts which are false as well. This unknown state space precludes the direct use of a planning model.

[4]Appelt (1981, 1985) also proposed planning models for conversational discourse. This work, however, principally concerned intra-utterance planning rather than inter-utterance conversational planning.

Power (1979) found that his stack-based planning system for conversation would run into problems because of (1) incompleteness and (2) insufficient flexiblity in adjusting to changes in context:

> Let us turn now to the second fault of the control stack as representation of the dialogue state: namely, insufficient explicitness. What this means is that the relations between elements of the dialogue state are not represented systematically.... The result is that the dialogue state can be interpreted just one way; it cannot be interpreted by several different procedures for several different purposes. The robots therefore cannot respond flexibly to unexpected turns in the conversation; an unexpected remark throws them completely. (Power, 1979, pp. 133-134).

To help solve this problem, Power proposed marking the elements of the dialogue state with explanatory relations that could be used inferentially to rework the plan. In effect, this would mean trying to replan the conversation at each state. This approach is not parsimonious and tries to graft skills for opportunism onto a fundamentally top-down structure. It does not really address the underlying issue of producing conversational organization *from* a system which is fundamentally flexible.

Cohen (1984), working with earlier research, reported that discourse analysis of human-computer interaction reveals that users do not follow the strict embedding of subdialogues required by an ATN model. Rather, a more flexible "demand" model was needed. Cohen also reports research indicating that efficiency in referential communication is a function of user feedback. ATN's, as a stack-based method, are considered too rigid for even sentential grammars and thus are unlikely to be capable of representing dialogue processes (Frederking, 1988).

Indeed, the planning model has been characterized as a post-hoc rationalization of actions; it is an artifact of reasoning about actions rather than a mechanism for producing them (Suchman, 1987). Plans, Suchman suggests, are simply a restatement of intention.[5] She proposes that the coherence of action is not adequately explained by either stored plans or scripts; rather, the organization of the conversants' actions is an emergent property of moment-by-moment interactions between actions, and between conversants and their context. That is, global coherence is the result of situated application of locally meaningful operators. This extends Birnbaum's (1986) notion of opportunistic planning

This does not preclude the use by conversants of high-level reasoning about their context and actions they might take to achieve their goals. This is, after all, a large component of what we perceive as conscious thought. Therefore, as I understand the implications of Suchman's thesis, we can produce conscious plans (based on some known state-space), act on them, and then react to changed or unanticipated circumstances as needed. More typically, we do not formulate an explicit plan; rather, we take some initial action to achieve our goal, and thereby create a set of expectations about what will follow. It does not matter (from the standpoint of conversational control) if the expectations are not met because we can again produce from our intentions a new action which is responsive to the new situation. To the extent that our expectations are met, we can routinize the selection and application of operators.

Summary

In this Chapter, I presented the integration problem: resolving speech-act theory with the ragged character of real conversation. Socio-linguistics has identified characteristics of linguistic interaction that (1) are difficult to explain in speech-act theory and (2) indicate

[5]Suchman also rejects speech-act models as begging the question of situated interpretation. As I attempt to show in this dissertation, speech acts and situated action are not incompatible. Speech acts can be viewed as the product of contextually sensitive operators.

the presence of meta-messages for conversational control. I then discussed techniques and assumptions which conversants use to maintain coherence. The evidence suggests that coherence is a product of a joint process in which the parties judge their understanding with respect to a mutual model of the conversation. When conversants detect significant differences between their beliefs about the mutual model of the conversation, they use feedback and repair to restore coherence. This process of feedback and repair can be viewed as consisting of meta-locutionary acts which control the interaction. As a result, the non-grammatical features of conversational interaction can be interpreted as embodying meta-locutionary acts which promote coherence. Finally, I discussed the role of intention in conversation, with particular attention to issues of interpreting expression and conversational structure. While the distinction between voluntary and involuntary linguistic action is not a clear one, a reasonable model of conversation will interpret displayed behaviors as acts. As to structure, the spontaneous nature of conversation suggests that planner-based models should be disfavored; global coherence may be produced through situated application of locally meaningful operators. In the next chapter, I more fully develop the idea of meta-locutionary acts as a theory of action in language.

CHAPTER III

A THEORY OF META-LOCUTIONARY ACTS

Approaches to the Integration Problem

If the descriptive requirements for speech acts are too restrictive, in what useful way might some requirements be relaxed? How could the resulting entities still be characterized as acts? One way to approach this problem is to enlarge the domain of the acts. Typically, speech acts concern the general domain of the conversation—the subject of the conversants' shared model. That is, traditionally defined speech acts have been applied to effectuating changes in the extra-conversational context, which is what I earlier called the "top" level of conversation. This means that speech acts have not been applied to actions concerning the conversation itself, or to human cognitive mechanisms such as recollection or focus of attention. As Cohen's (1984) work made clear, Austinian speech acts cannot adequately explain attention-focusing utterances independent of some act of predication. This failure does not invalidate the theory of the process of generation and comprehension of language which is implicit in speech-act theory. Under this theory, speakers are goal-directed independent agents whose instrumentalities for achieving their goals include language acts. Recall that Austin's original idea was to explain how people did extra-conversational things with words. As I earlier suggested, this idea can as well be applied to things which people do which are intra- or meta-conversational. Thus as conversants through interactive discourse create a shared model of their conversation, they undertake a sequence of acts which effect changes in the model by (1) making acts within the directly copresent mutual knowledge of the conversants and (2) understanding the effects of these acts on the other conversants.

The conversants share responsibility for the maintenance of the model. Note that here I am referring specifically to the conversants' model of their own interaction rather than a model of their mutual beliefs about the subject of the conversation. Jernudd and Thuan (1983) characterized the behavior of conversants in this way, noting that the social aspects of language provide a set of *shields* or protocols which allow conversants (and here the emphasis is of course on hearers) to employ a wide variety of correction (i.e., maintenance) resources. These protocols, they stated, allow for shared responsibility for speaking. Clark and Wilkes-Gibbs independently proposed a *principle of mutual responsibility*:

> The participants in a conversation try to establish, roughly by initiation of each new contribution, the mutual belief that the listeners have understood what the speaker meant in the last utterance to a criterion sufficient for current purposes. (Clark & Wilkes-Gibbs, 1986, p.33)

Clark and Wilkes-Gibbs described the operation of the principle of mutual responsibility in the case of definite reference but, as Jernudd and Thuan suggested, such a principle is appropriate for most aspects of interactive discourse. For definite reference, this process involves initiation and iterative expansion, replacement, or repair until the conversants arrive at a version of the reference they mutually accept. The conversants try to minimize collaborative effort. Thus if the speaker utters an elementary noun phrase the hearers can presuppose their acceptance without taking an extra turn. Clark and Wilkes-Gibbs noted Schegloff's finding that speakers are usually allowed to present utterances without interruption; repair communications come in the interstices between utterances. Schegloff et al. (1977) in fact stated that other-initiation is withheld in order to allow speakers an opportunity to initiate repair themselves. Clark and Wilkes-Gibbs (1966) then observed that this explains why allowing a new contribution to proceed is tantamount to a mutual acceptance of the old one.

Cohen's (1984) approach to this problem was to separate the goals of reference and predication and to satisfy them separately. He suggested that in the case of apparent predication without reference the solution is to understand the utterance as an indirect speech act of reference. In either case, there has to be an action of referent identification. Thus Cohen argued that referent identification should be treated as an action that speakers request, and that the speech act of referring should be considered a kind of request. In all three instances (Jernudd & Thuan, 1983; Clark & Wilkes-Gibbs, 1986; Cohen, 1984), the solution to the integration problem comes through applying speech-act-like analysis to various intra-conversational aspects of discourse. They in effect break up speech acts to cover the particulars of their respective problems studied. The theory of meta-locutionary acts presented in this dissertation extrapolates this method to a generalized approach to interactive discourse.

Although the extension of speech-acts to meta-communication is novel, the idea of meta-communication as a factor accounting for coherence has been the subject of at least preliminary research. It appears that the choice of a direct speech act for embodying an indirect act is, at least in part, a function of social relationship. The form of act then conveys as meta-communication the nominal social statuses of the conversants (Sanford & Roach, 1987). Conversants can then use this meta-information to interpret indirect speech acts. Note, though, that this approach does not involve relaxing constraints of Austinian speech acts. Rather, Sanford and Roach are describing the *choice* of Austinian acts as conveying meta-information. For conversants, this is part of the process of reasoning about acts.

There is some empirical evidence in support of techniques which break down domain-level speech acts. Schegloff et al. (1977) found that where one conversant initiates repair of another's utterance, the operations of locating the repairable discourse units and supplying a candidate repair are separated. The techniques for initiating a repair of another's utterance are technique for locating the "trouble source." In this case the actions of maintaining the shared model of the conversation can be broken down at least into actions which shift the (sub-) focus to a problem with the model and actions which repair the problem.

A Computational Model of Meta-Locutionary Acts

In the previous discussion I have reviewed the fundamental reasons for computational models of interactive discourse which follow the successful route of speech-act theory but which relax the theory by applying it to the shared model of the conversation itself. This is the consequence of the principle of shared responsibility. However, the discussion so far has been synthetic, in that it has attempted to extrapolate from various approaches to the integration problem yet has largely remained in the realm of motivation rather than extension. Here, then, the discussion becomes developmental. In this section, I outline the areas in which computational models of interactive discourse can be used to analyze this problem.

Requirements for a Model

The search for a meta-locutionary model does not begin in a vacuum. Schegloff and others have shown how repair has structure. This structure is not coextensive with chunks of linguistic expressions such as lexical items, intonations, or syntactic constructions. Rather, as I understand the import of this work, it is the relationship among these chunks which creates a repair structure through the local dynamics of the communication itself (Jernudd & Thuan, 1983; Suchman, 1987).

One way in which the mechanism of the feedback process has been examined is through timing and coördination of correction utterances. (This is a sort of meta-coördination issue.) The socio-linguistic research examining how people direct the course of conversation and repair its inherent troubles has shown that conversants manage who is to talk at which times through an intricate system of turn-taking. It turns out that there is a pattern and structure to initiation of repairs which can be described in terms of domain-level turn-taking:

> The 'repair-initiation opportunity space' is continuous and discretely bounded, composed of initiation-opportunity positions at least some of which are discretely bounded. The positions are adjacent, each being directly succeeded by a next, some being themselves composed internally of a set of 'sub-positions'. [Footnote omitted.] (Schegloff et al., 1977, p. 375)

I expect that instances of positive feedback will fall neatly into these positions as well, indicating prophylactically to the speaker that the shared model is consistent thus far.

Feedback in interactive discourse displays other procedural regularities. Schegloff et al. (1977) noted that repairs are subject to two strong preferences: speakers prefer to repair their own utterances rather than let hearers do it; and speakers prefer to initiate their own repairs rather than let hearers prompt them to do it. As a result, conversants repair their own utterances as soon as they detect problems. This is consistent with the results of Clark and Wilkes-Gibbs (1986), who found that speakers prefer to make their own expansions of noun phrases without prompting. There are places in the conversational flow which the conversants mutually recognize in which repair initiation ought to occur if repair were needed. A place-marking act which does not initiate repair can then be understood as indicating comprehension (Suchman, 1987).

At this point I want to interject, though, some methodological perspective on these results. From the published research, it appears that Schegloff et al. studied transcripts of verbal interaction. These transcripts at best coded pause, intonation, and audible breath; although "non-linguistic" behavior such as gesture may be parenthetically noted, they may have missed other extra-linguistic aspects of discourse such as gaze. As a result, the only attempts at correction which are reflected in the research analysis are verbal ones. It may be, and preliminary protocol analysis of my own so suggests, that nonverbal (but nonetheless linguistic) communication may contain a significant level of correction, repair, and other feedback information. As a consequence, other-initiated repairs may be underreported and the preference for self-repair correspondingly over-reported. To the extent which they are true, though, do the reported aspects of the process have any advantage for interactive discourse? Clark and Wilkes-Gibbs (1986) observed that conversants also try to avert potential exchanges as the hearer tries to correct any misunderstanding; that is, the conversants strive to achieve mutual acceptance of a reference with minimal conversational cost. These preferences have the effect, then, of minimizing collaborative effort. Accordingly, we can develop a number of standards by which computational models of feedback in discourse can be evaluated. Is the model a performance model with respect to repair opportunities? Does the model inherently serve to minimize collaborative effort?

Levels of Feedback for Modeling

There are differences among kinds of feedback, in the sense of supportive versus corrective, or self-repair versus other-initiated repairs. Goodman (1986) found four kinds of communicative failures in the water-pump assembly conversations he studied: erroneous specificity, improper focus, wrong context, and bad analogy. There are also differences in the level of the feedback with respect to the multi-stratification of conversation. Some corrections and assurances are obvious because they are embodied in sentence-level utterances. For example, Grosz reported the following fragment:

S1: The lid is attached to the container with four 1/4-inch bolts.

S2: Where are the bolts? (Grosz, 1977, p. 353)

Very typical of the dialogues reported by Cohen is this exchange:

S: Take that red piece. It's got four little feet on it.

J: Yeah.

S: And put the small end into that hole on the air tube—on the big tube.

J: On the very bottom.

S: On the bottom, yes.

J: Okay. (Cohen, 1984, p. 140)

In the first example, the request for clarification is at the extra-conversational level of Austinian speech acts. In the second exchange, we begin to see fragments: utterances which cannot be coded directly as speech acts. This exchange, even though purely textual, shows the beginning of the journey down through the layers of the conversational structure; it can still be coded reasonably through a minimally relaxed set of speech acts. There are some kinds of perlocutionary effects, though, which can be achieved linguistically, but not through communication of intent (Cohen, 1984). These are farther down in the structure. It is not clear where the bottom is. Do conversants recognize intent for fundamentally intra-conversational acts? Actually, this question is not settled even for domain-level acts (cf., Allen & Perrault, 1980). Recognition, as with other conversational processes, need not be conscious. At some point, though, we will reach another set of not-usually-negotiated communicative units through which the collaborative structure is created.

Forms of Computational Models

The goal of this research is to identify and to model computationally a set of acts relative to the conversational levels. The set of acts which we choose as a representation for the real acts in the world of actual conversants depends on the computational structures within which the acts are used in the processes of comprehension and generation. Is the process simply pattern-matching? Parsing? Inferential?

Clark and Wilkes-Gibbs (1986) stressed that the principle of mutual responsibility operates in a context of widespread, natural ambiguity and uncertainty. Conversants need not assure perfect understanding of each utterance but only understanding to a criterion sufficient under the circumstances. The dynamics of the collaborative process encourage minimizing the collaborative effort, thus implicitly encouraging some measure of uncertainty in understanding. Certainly, hearers will accept uncertainty when they anticipate that the gaps in their understanding will be filled in later. But, as Clark and Wilkes-Gibbs pointed out, the hearer has to tolerate uncertainty always: Although conversants might like to accept each element mutually, second by second as they proceed, this represents an impractical ideal. First, there is the probability that some definite references cannot be understood fully until later in the utterance. Second, I note that each part of the utterance itself is a case of the speaker being ahead of the hearer. This is exactly what Grosz (1981) described as the speaker always being one step ahead of the hearer. This suggests two further points about the nature of feedback itself: First, no matter how the observed discourse units are subdivided or aggregated (and they always can be), the speaker is always this unit ahead of the hearer and both speaker and hearer may be depending on the hearer's feedback to maintain their model of the conversation. Second, the simultaneous presence of discourse segments, units, fragments, and atoms of various sizes, almost always overlapping of others, suggests that feedback with respect to the mutual model of the conversation must itself be multi-layered, with related temporally and referentially dependent communicative functions. Adequate computational models of interactive discourse, then, must reflect this multi-layered, simultaneous character.

Meta-Locutionary Acts

To determine a set of illocutionary meta-acts, I start with the taxonomy of acts proposed by Searle (1969) that form the foundation of speech-act theory. Searle suggested that the types of illocutionary acts are promise, request, assert (or affirm), question, thank, advise, warn, greet, and congratulate. For each type, Searle defines the act's propositional content, the conditions preparatory to its use, the intent (which Searle calls the sincerity) of the speaker, and the act's essential meaning. Can these acts be applied directly in a meta-locutionary way?

The Role of Acts in Language

Some of Searle's acts do not have obvious analogs for the control processes of conversation. For example, greeting and congratulating seem inherently tied to the domain of social relations. The mere existence of other acts might be questioned. What, after all, are acts in language? The easiest act to understand is a request. Requests have an obvious motivation for the speaker; namely, a goal that the hearer perform the requested act. But in what sense is an assertion an act? Searle says that a "propositional act" can be a constituent part of an illocutionary act of asserting. The apparent, surface motivation for assertions is a goal that the hearer know something. Yet is it ever the case that the speaker really wants as an end in itself that the hearer should know something? Typically, one might surmise, the speaker really wants the hearer to know something because the speaker believes that this will bring about some further, elaborated state which is the speaker's greater goal. If a child wants someone to bring it a drink, the child might say "Hey, I'm thirsty." This is an act of assertion which has as its immediate goal the state that the hearer should know of the speaker's thirst. It can further be seen as act of requesting, where the speaker's goal is to get the hearer to bring the speaker a drink. This case is a clear example of an indirect speech act. Thus, one can ask, is an assertion ever an act in itself? Actually, examples of assertion are not hard to find. A tutor's statement, under the usual circumstances, that "The capital of Delaware is Dover." is an act of assertion that has as its intended effect the acceptance of the content of the statement by the tutor's student.[6] The student's might in return say "OK," thus making an assertion that she understands the tutor's statement. In the context of the control of conversations, an act analogous to the student's confirmation would be a conversant's acknowledgment that it was the other conversant's turn to speak.

Domain-Specific Acts

If meta-locutionary acts are the extension of illocutionary acts to subsentential, meta-locutionary phenomena in conversation, then are these acts the same acts which are used at the sentential, domain level? One possibility is that the set of acts is the same, and that the instantiation of the acts with meta-locutionary states differentiates them from acts instantiated with domain states. The other possibility is that the acts are different in themselves. That is, there is one set of acts which is peculiar to meta-locutionary phenomena and another set for domain knowledge. In fact, this position might even be extended by the idea that every different domain has a different set of acts and that

[6]The fact that the tutor's act may be a pure assertion does not imply that the tutor lacks more complex intentions which are served by the act. The tutor may have intentions, for example, that the student will pass an examination. However, the tutor cannot achieve this directly through an act. Rather, the tutor—through assertion—takes an act which has a change in the mental state of the student as its immediately intended effect.

meta-locutionary phenomena simply comprise a particular, recurrent domain. The first position I will call universalist; the latter situated, in the spirit of situated-action analysis (Suchman, 1987).

The existence of different sets of acts for different domains postulated in the situational position is not as bleak a representational prospect as might be imagined. Specialization of acts does not negate the idea of ontological structure for classes of acts. Just as we can apply old words in new domains, or understand new words in old domains, so too can we interpret unfamiliar acts. It may also be the case that acts are reified by specialized use. In other words, the constant use of particular (perhaps partial) instantiations of certain acts may lead to general acceptance of these instantiations as acts in and of themselves. Jernudd and Thuan (1983) pointed out that speech acts and their meanings may be negotiated between conversants. Of course, a huge number of pre-negotiated acts is part of the language as we learn it. As a consequence, one would expect to find the greatest levels of specialization in the domains which we most frequently use. If the process of conversation is itself considered as a domain, then the use of specialized meta-locutionary operators should be highly developed.

Note that ontological relationships should still hold among acts which are specialized for the meta-domain of conversational control. Suchman's (1987) situated action hypothesis, that conversants have highly developed operators which respond to local context, also leads to the conclusion that the acts used in the experimental domain may fairly be represented as domain-specialized acts. Indeed, meta-locutionary acts ought to be derivable as instantiations of more basic acts. Accordingly, at the turn-taking level, I suggest that there are specialized acts which take into account the particulars of these behaviors. These acts include acknowledge-turn, give-turn, request-turn, and hold turn.

Each act should, in theory, be derivable from some set of more general acts. For example, request-turn could be expressed as

act (Me, request (Other, act (Other assert (belief (turn (Me)))).

The concept of turn itself might be then be explicated further. The point here, though, is that there are domain-specific acts which have an ontological relationship to a more abstract set of acts upon which conversants might rely in an unfamiliar context. Givón (1984) observed that as languages develop, they grow gradually less ambiguous and more richly coded. That is, in their early stages languages tend to be highly polysemous. As linguistic functions become routinized, specialized functionality develops. In the context of speech acts, then, the general acts identified by Searle represent the ontological primitives from which the specialized meta-acts developed as the functions they facilitated became routine.

A form of taxonomy of meta-locutionary acts can be developed using the general classifications of speech acts as an outline. More precisely, there is a forest of taxonomies where each tree corresponds to a conversational level. I will follow the taxonomic outline proposed by Bach and Harnish (1979) for communicative illocutionary acts.[7] *Constatives* express the conversant's belief and her intention that the other conversant have or form a like belief. *Directives* express the conversant's attitude toward some prospective action by the other conversant and the first conversant's intention that her utterance, or the attitude it expresses, be taken as a reason for the second conversant's action. *Commissives* express the conversant's intention and belief that her utterance obligates her to do something (perhaps under certain conditions). *Acknowledgments* express feelings regarding the other

[7]In addition to communicative acts, Bach and Harnish (1979) also described "conventional" illocutionary acts, which are not relevant to the theory developed in this dissertation. Briefly, conventional illocutionary acts include effectives and verdictives, which mainly concern socially conventional acts such as vetoing a bill or finding a defendant guilty.

conversant or convey formal or perfunctory expressions that social obligations be met. The use of Bach and Harnish's particular taxonomic system is intended as an explanatory aid; it is the identification of (and the relationships among) the various meta-locutionary acts which matter most. Using these distinctions then, Figures 3, 4, 5, and 6 present taxonomic trees for certain meta-locutionary levels of the conversational process; respectively, these are turn-taking, repair of mutual models, reference/information, and attention.

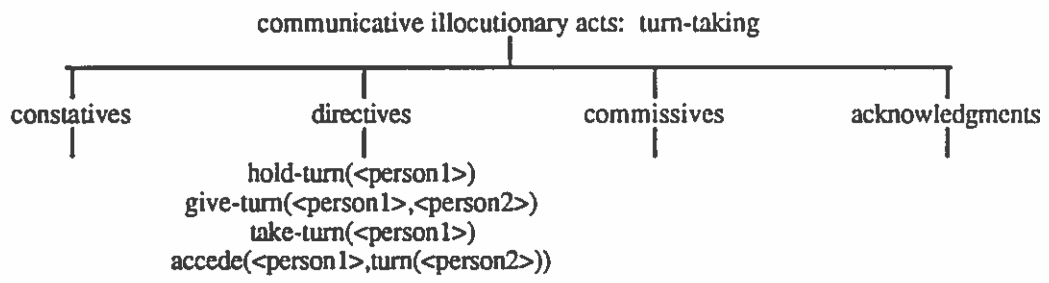
communicative illocutionary acts: turn-taking
constatives
directives
commissives
acknowledgments
hold-turn(<person1>)
give-turn(<person1>,<person2>)
take-turn(<person1>)
accede(<person1>,turn(<person2>))

FIGURE 3. Taxonomy of meta-locutionary acts for turn-taking.

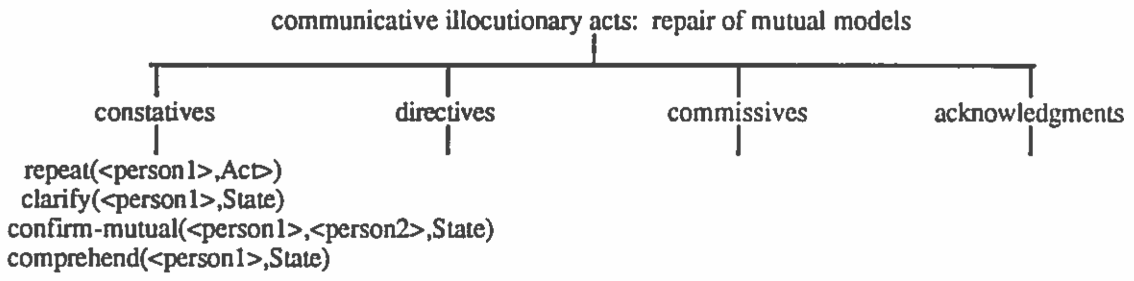
communicative illocutionary acts: repair of mutual models
constatives
directives
commissives
acknowledgments
repeat(<person1>,Act>)
clarify(<person1>,State)
confirm-mutual(<person1>,<person2>,State)
comprehend(<person1>,State)

FIGURE 4. Taxonomy of meta-locutionary acts for repair of mutual models.

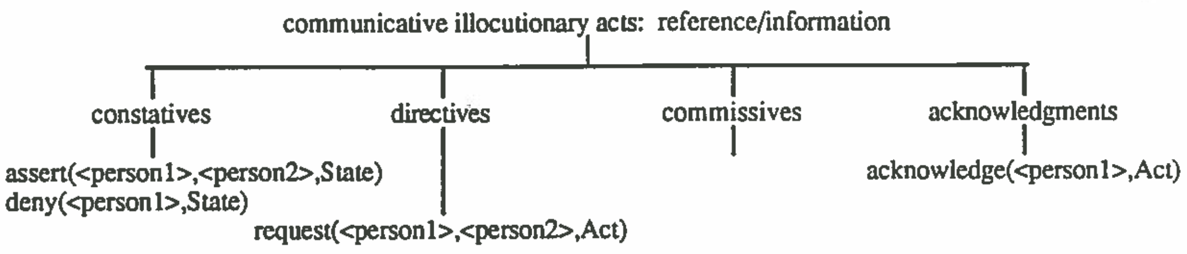
communicative illocutionary acts: reference/information
constatives
directives
commissives
acknowledgments
assert(<person1>,<person2>,State)
deny(<person1>,State)
request(<person1>,<person2>,Act)
acknowledge(<person1>,Act)

FIGURE 5. Taxonomy of meta-locutionary acts for information.

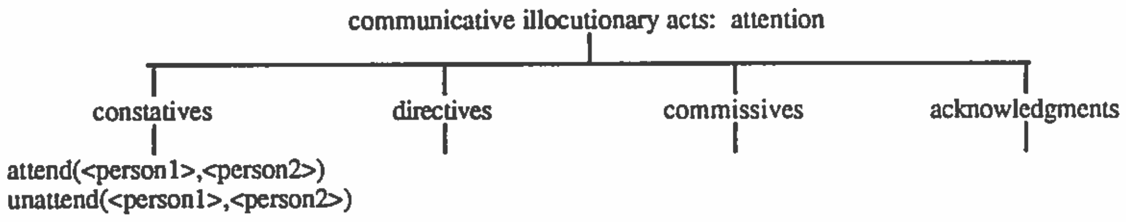
communicative illocutionary acts: attention
constatives
directives
commissives
acknowledgments
attend(<person1>,<person2>)
unattend(<person1>,<person2>)

FIGURE 6. Taxonomy of meta-locutionary acts for attention.

Summary

In this chapter, I presented the foundations and elaboration of a theory of meta-locutionary acts. The theory arises from the principle of mutual responsibility in conversation, which holds that conversants share the responsibility for maintaining the coherence of their interaction. This implies, among other things, that conversants must have means for confirming copresence and the effects of conversational acts on the other conversants. These control functions can be modeled as speech acts through relaxation of constraints on Austinian speech acts. In particular, going beyond the mapping of speech acts to domain (i.e., extra-conversational) matters permits the acts to reference and affect the conversation itself; that is, for these acts, the conversation *is* the domain. Such meta-locutionary acts, then, are reflected in a computational model which proposes to encompass the processes of repair and feedback which maintain the conversational models. These acts cross conversational levels and are simultaneous. Finally, I presented a taxonomic account of the meta-locutionary acts, with particular application to turn-taking, repair of mutual models, information, and attention.

Given the theory, though, two questions arise. First, does the theory correspond to and account for the interaction observed in actual conversations? Second, is the theory adequate for computer-based interaction? The question of the explanatory power is addressed in Chapters IV and V. The question of the applicability of the theory is addressed by the computational simulation in Chapter VI.

CHAPTER IV

METHODOLOGY

In the preceding chapters I considered that the general question of coherence is posed in the integration problem, examined a range of prior work which provided foundations for attacking the problem, and proposed a theory of meta-locutionary acts for explaining the control processes of conversation. In this chapter I discuss the specifics of a study for assessing the theory. In particular, I look at the effects to be studied, and the mechanics of the protocol analysis of conversation. The computational modeling of the observed behaviors is discussed in chapters V and VI.

Methodology in Cognitive Science

Although the field is not settled (Suppes, 1984), methodology in cognitive science has produced some agreement on what constitutes adequate work. Unfortunately, research into the processes of conversation has (at least) two major methodological pitfalls. The first is the lure of introspection. Introspection seems inexpensive and easy; virtually every human being communicates interactively with others. But one cannot simply present one's intuitions as a descriptive study. It is precisely because mixed-initiative discourse is so familiar to us that its processes are transparent to the introspective observer. The observed behaviors must be explained in the context of a theory that proposes a coherent model (Miller, Polson, & Kintsch, 1984). What sort of theory is adequate? Miller, et al., (1984) suggest that there is a continuum between a naked idea and an explicit process model. In the center of this continuum lie "middle ground" theories in which structures and processes have been specified in sufficient detail to support both the instantiation of the theory as a computer program and qualitative experimental study. In this chapter, I explain how the theory of meta-locutionary acts can be validated in both regards; it thus avoids the first pitfall. The second pitfall is that of failure to explain behavior as a process. Although all studies must inherently look back at behavior that is now—because it occurred in the past—static, a good model will explain the behavior on the basis of real-time factors: those factors which were available in the actual context of the observed interaction (Swinney, 1984). Although this view has been criticized as overly restrictive (Charniak, 1984), its essential truth is that the model cannot embody in its explanation the kinds of post-perceptual processes which are available to observers rather than to the participants. I have tried to address this concern through (1) minimizing domain-level, real-world knowledge in the experimental task, (2) using temporally-significant acts to explain the observed behaviors, and (3) trying to replicate the behaviors with a simulation of the model.

Effects to be Studied

The eventual goal of this study is to test the theory of meta-locution through computational modeling of actual interactive discourse. Within the general philosophical framework of the speech-act approach to language, most of the discourse theories we have discussed have looked at basically sentence-level (or utterance-level) interaction. At a minimum, the discourse unit of interest has been the noun phrase, where the noun phrase has been characterized as a act of assertion or the subject of negotiation. As I have tried to show in Chapter II, both Cohen (1984) and Clark and Wilkes-Gibbs (1986) took constructive steps toward solving the integration problem through partial reduction of the units of discourse. Taking this research as leads or indications of the proper direction for more general solutions of the integration problem, one can go much farther in atomizing discourse for purposes of both understanding and generation. In Chapter V, I extend the analytical techniques such as those used by Grosz, Allen, Clark, and Cohen to sub-domain levels of discourse.

I thus seek to examine sub-domain levels of conversational interaction. As the socio-linguistic literature has described the features of conversation which apparently lead to coherence, this study concentrates on aspects of language which are related to these observations. Thus the model should explain turn-taking, negotiation of reference, and confirmation of the mutuality of knowledge.

General Approach

I have argued that maintenance of a conversational model is clearly part of any reasonably sophisticated approach to generational simulation of conversation, yet our initial observations of conversations suggest that maintenance of the model is a continuous, multi-level process of incremental addition and revision rather than a post-utterance assessment. This set of behaviors of the speaker and hearer is (1) at a more finely grained level than that of the standard illocutionary act and (2) about the process of conversation itself. They are, broadly speaking, locutions which embody illocutionary acts in the sense that they are intended to produce a change of state in the world outside the speaker, and are meta-acts in the sense that in combination they effect Austinian speech acts but individually are specifically directed to changes in the state of the conversational model itself. That is, these are illocutionary acts which correspond to intended perlocutionary effects performed on the shared conversational structure. The feedback-suffused basis of coherence in conversation is a critical fact (VanLehn, Brown, & Greeno, 1984) which the theory of meta-locutionary acts should better explain than traditional speech-act theory.

Given a theory of the meta-(il)locutionary act as a maintenance device for a multi-layered shared conversational model, what is the immediate program of research to be followed? Cohen (1984) states that one should derive the illocutionary acts as a rational strategy of action, given attributions of participants' beliefs, goals, and expectations at the point in the discourse in which the illocutionary acts actually occurred. Accordingly, I first identify a set of speech acts which (1) comprise illocutionary acts and (2) handle the micro-tasks of turn-taking, negotiation of reference, and confirmation of the mutuality of knowledge. This is accomplished through analysis of conversation protocols at a sufficiently small granularity that the acts can be discerned. Second, I develop, using these acts, a computational model of the belief structures of the conversants. The structures, with a suitable set of operators, are sufficient to account for most of the observed meta-locutionary behavior.

Study of Spoken, Face-to-Face Conversation

The effects of meta-locutions, as I have defined them, are peculiar to real-time interaction and especially to spoken conversation. However, not all discourse is interactive or spoken, much less face-to-face. In fact, there are a number of dimensions which characterize the various modalities of discourse. The character of the discourse changes in ways corresponding to the characteristics of the modality.

Why then should one conduct research into computational models of meta-locutions using spoken, face-to-face conversation? It appears that the most efficient (in temporal terms, anyway) form of interactive communication for human beings is spoken conversation. It is certainly easier. Why should users of interactive computer systems be limited to the capabilities of computing technology circa 1965? Moreover, it may be difficult or impossible to understand the processes of lanugage and interaction without starting with face-to-face, spoken conversation. While the precise nature of the contributions of verbal and nonverbal communication to interaction are not known, they are ascertainable through research:

> This notion, though simple and intuitive, carries a strong implication for research: we can fully understand language only by examining its functioning as an aspect of face-to-face interaction, and not be treating it as an autonomous entity, unsullied by contact with everyday social processes, including co-occurring "nonverbal" actions. (Duncan, 1980, p. 67).

Looking at the advantages of conversation to interfaces, cross-modal research on discourse as surveyed by Cohen (1984) found significant efficiencies in spoken conversation. In particular, a series of cross-modality studies was conducted by Chapanis, Ochsman, Parrish, & Weeks (1972, 1977). They found, inter alia, that problems are solved twice as fast in vocal modalities as they are in written ones, even though conversants use twice as many words when speaking. Cohen (1984) also cited a thread of psychological research on cross-modal comparisons of reference. These studies showed that for spoken interaction the length of noun phrases tends to decrease as subsequent references are made; this decrease is not as sharp for non-interactive spoken modalities. Cohen concluded that these results indicate that efficiency in referential communication is a function of conversants' feedback.

Many of the differences between narrative and interactive discourse are well known. People looking at conversational language often remark on its apparently ill-formed and "ungrammatical" qualities. Not only are what one often considers speakers' mistakes more prevalent, correspondingly abundant are the opportunities for repair. Thus Goodman (1986) suggested that the study of miscommunication is a necessary task for building natural language understanding systems since any computer capable of communicating with humans in natural language must be tolerant of the complex, imprecise, or ill-devised utterances that people often use. Interactive systems which aspire to live in the real world of feral language must be able to cope with its perplexing characteristics. Among the factors affected by modality of discourse is the conversants' ability or opportunity to maintain their model of the conversation. Clearly, authors of novels do not depend on real-time feedback from their readers to check uncertainty in the readers' models, nor can authors negotiate the meaning of acts, lexical items, and references. Thus Jernudd and Thuan (1983) observed that discourse represents a continuous process of accommodation among conversants.

The relationship between conversants can vary greatly, from the great distance of written discourse to closeness of face-to-face conversation, which brings the question of accommodation into sharp focus (Jernudd & Thuan, 1983). If one is to study the shared model created by discourse and its maintenance through negotiation and accommodation, then one should study conversational rather than textual discourse. Other modalities of discourse, such as keyboard input, can be interactive but not spoken. It turns out that the presence of speech itself in place of written interactive language is significant for the structure and processes of discourse. Cohen (1984) found that keyboard communication is distinctly different from other modalities of discourse like face-to-face conversation. In particular, among other differences, keyboard interaction emphasized optimal packing of information into the smallest linguistic space. As a result, keyboard communication alters the normal organization of discourse. Of course, non-textual information represents some sort of (presumably most efficient) lexicalization of an underlying meta-locutionary act. To the extent that acts can be lexicalized in the alternate modality, the meta-locutionary content can be transmitted. However, this may require a new vocabulary; unless recognized though social acceptance or immediately negotiated, new lexemes will not be understood.[8]

Modality-induced differences in discourse also include the way conversants use reference. For example, voice-only communication removes some of the forms of acts of reference (e.g., deixis and common visual context) which keep the interaction coherent. When these acts are not available, the interaction can easily break down:

> O.K., uh ... now, we need to attach the um ... conduit to the motor ... the conduit is the uh ... the covering around the wire that you ... uh ... were working with earlier. Um, there is a small part um ... oh brother ... (Grosz, 1982, p. 88)

Accordingly, cross-modal studies have shown that different modalities of communication lead to different uses of referring expressions. Analysis of protocols from teletype and telephone interactions shows marked differences in the use of explicit requests for identification. As a consequence, systems which understand spoken language will have to be prepared for language which differs from that observed in teletype interaction (Cohen, 1981). If the goal of research in natural language processing is either (1) understanding how people actually converse or (2) developing systems which converse with people, the evidence with respect to differences in conversational characteristics from differences in modality suggests that spoken conversation would be a fruitful area of study.

The Study

The methodological strategy of this study, then, is to examine conversational interaction in some reasonable domain, and then to derive the underlying illocutionary acts as a rational strategy of action, given attributions of the participants' beliefs, goals, and

[8]Some new lexemes may be slowly evolving for textually based computer interaction. For examples the smiley-face icon ":-)" and its variants convey the meta-locutionary sense of satire or irony.

expectations at the points in the discourse in which the illocutionary acts actually occurred (Cohen, 1984). I thus turn to the particulars of the design of the study that instantiate this strategy.

Domain and Tasks

The first part of the empirical work involved development of a suitable domain for the observed conversations. The general requirements were that the conversation produce acts by the conversants which included turn-taking, negotiation of reference, and determination of mutuality of knowledge. In order to facilitate ready experimental determination of mutuality, the domain should be a simplified one in which as much of the mutuality as possible is created through direct rather than indirect copresence. That is, if the knowledge becomes mutual to the conversants during the conversation itself, the fact of the mutuality can be more easily ascertained by an experimenter-observer. In contrast, if the subject of the conversation is something which is mutual through indirect copresence, then the experimenter has virtually no basis from which to determine the knowledge confirmed or the extent of the confirmation. Unfortunately, extreme reduction in domain complexity may also lead to unrealistic interaction that exhibits artificial effects. In reaching a balance here, one may simply have to qualitatively evaluate the resulting conversation for verisimilitude.

A second factor in selecting the domain concerns deixis. I sought to reduce confusion over gestures which were referential by minimizing circumstances which called for deictic reference. This meant that a task such as jointly building a structure would have to be limited to structures which were wholly mental rather than physical or visual. However, the advantages of reduced confusion from deictic gestures are not obtained without cost. DeLancey (personal communication, February 23, 1988) has pointed out that a shared physical work serves as a default place for gaze; thus where the conversants have something other than each other to look at, direction of gaze toward each other may be more significant than in the case where they look at each other more or less continuously.

A third factor which influenced the choice of both the domain and the task was a need to keep the conversations reasonably short—around two minutes at the most. At the fine-grained level of transcription which this study requires, longer protocols would simply be impractical to transcribe. Experience with a pilot study suggested that to transcribe fully ten seconds at the appropriate level of detail takes about six hours. Thus forty seconds of protocol requires three working days to transcribe. It would be also be possible to elicit longer conversations with greater structure and then choose short sections for analysis. However, this would lead to more speculative analysis in initializing the conversational models. Accordingly for this initial study, the domain and task had to revolve around concepts and activities that could have fairly rapid closure.

Given these constraints, I chose for the domain the task of jointly reconstructing a sequence of random letters. Thus a typical protocol involves two subjects, each of whom has a copy of the same sequence of 15 letters. Some of the letters have been replaced by blanks. The blanks do not overlap. Thus if one had both copies of the sequence, the entire sequence could be determined. The sequences of letters and the positions of the blanks were chosen randomly. Figure 7 depicts the copies of the sequences which subjects received in one of the trials.

The tasks were structured as follows: The subjects were given, by random choice, their copies of a sequence. The subjects then had one minute in which to memorize their respective sequences. After the minute expired, the subjects turned over the pieces of paper on which the sequences were printed and were instructed to recall jointly the entire sequence. This task was repeated two additional times, so that for each pair of subjects I obtained three conversations. In the third task, the copies of the sequence were altered so that toward the end of the sequence the copies differed by one letter. Copies of this kind are depicted in Figure 8.

Copy of sequence given to subject A:

```
_ i s u t w r q g l d _ _ _ _ g o
```

Copy of sequence given to subject B:

```
o _ s u t w r q g _ _ f w w d g o
```

FIGURE 7. Example of random-letter sequences used in protocols. Neither copy contains the entire sequence. The blanks do not overlap. Therefore, the entire sequence can be reconstructed if the information in both copies is used.

Copy of sequence given to subject A:

```
n _ _ b w _ e g u q t y v o x u e
```

Copy of sequence given to subject B:

```
_ i k b w f e g u q t y _ _ _ y e
```

FIGURE 8. Example of non-identical sequences used in third task. The copies of the sequences give to the subjects differ toward the end. In this example, the difference occurs in the next-to-last position. This stimulus was used to induce apparent failure of the subjects' models of the conversation.

The rationale for the three tasks is that the first provides familiarization to the subjects, the second lets the subjects interact proficiently, and the third uses the subjects' expectations garnered in the first two tasks to induce apparent failure of the subjects' model of the conversation. The domain has the needed characteristic that all of the knowledge will be directly copresent, because the entire domain-level knowledge structure is contained in the sequences. Moreover, because the sequences are memorized, the resulting conversations revolve entirely around mental rather than physical constructions. The relatively short length of the sequences meant both (1) that memorization and recall would be feasible tasks and (2) that the conversations could conclude rapidly. These assumptions were confirmed in a series of pre-experimental tests.

Protocols

Protocols were obtained from two sets of subjects. Each set consisted of two persons who performed the three tasks described above. The subjects sat roughly at a ninety-degree angle to each other so as to face each other and still be visible to the camera. An overhead view of the situation is presented diagramatically in Figure 9.

The subjects were recorded on half-inch VHS videotape during all instructions, preparations, and the actual tasks. The subjects completed, from their subjective standpoint, all of the tasks, although objectively they did fail on occasion to recall the sequences accurately. However, their accuracy in the task was sufficient for the analysis. The conversations typically lasted about two to three minutes. In all, then, I obtained six protocols.

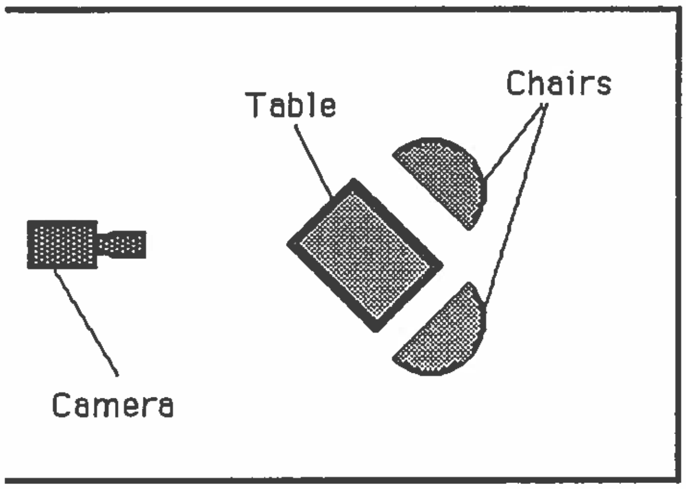

FIGURE 9. Overhead view of the experimental set-up for the letter-sequence protocols.

Protocol Analysis of Conversational Interaction

In the preceding section, I discussed the rationale, design, and collection of the protocol data for this research. I now turn to the analytical methods applied to the protocols. Useful sections of the protocols were transcribed for both verbal and non-verbal behaviors. The verbal portions were transcribed with the aid of the SoundCap™ digitizing program on an Apple Macintosh computer. The sounds were resolved down to a precision of about a tenth of a second. Ambiguous words were resolved by listening to the original videotape. Sounds that could not be resolved into words were noted phonetically. Intonation was coded through punctuation. The subjects' physical actions were transcribed using an editing videotape recorder with slow-motion control. The actions were resolved, where necessary, on a frame-by-frame basis down to a thirtieth of a second. A notation for the physical actions was developed to the extent necessary to impart the general nature of the actions observed; while accuracy in timing is important, accuracy in the specifics of the motions is not needed if the gist of the action can be discerned. I used a partial adaptation of the notation developed by Birdwhistell (1970). This feature of the transcription process was facilitated by the fact that the study does not examine nor does it make claims about lexicalization of physical action (or of verbal acts either, for that matter). A transcript of the particular protocol analyzed in this dissertation is set forth in Appendix A.

Once the verbal and physical behaviors had been transcribed, I then assigned identifiers to the behaviors which I deemed could be considered acts. These included virtually all of the behaviors observed. Only later in the study were behaviors classified as acts; the motivation for this approach is that it was better to over-include acts as significant rather than risk ignoring a behavior which might have been a significant within the context of the conversation. As Birdwhistell (1970) observed, no behavior ever carries meaning in and of itself; it is the context which provides the meaning, if any. For example, in the protocol reported in Appendix A, B's behavior of leaning back was initially coded as a separate act identified as Bi3; subsequent analysis of the interaction suggested no reasonable basis for finding this action as an act distinct from its other associated actions, and so the notation of Bi3 was deleted. As it turned out, though, very few behaviors were found insignificant. Conversely, in some cases additional acts were noted where, on further analysis, the behaviors appeared to require decomposition. In the protocol reported in Appendix A, act Ai6a was added in this way.

Summary

In assessing the validity of the theory of meta-locutionary acts, the effects to be studied should include turn-taking, negotiation of reference, and confirmation of the mutuality of knowledge. Accordingly, a domain was developed in which these meta-conversational phenomena could be observed and modeled. The domain task required face-to-face, spoken conversation between the conversants, maximized the use of directly copresent knowledge, and minimized deictic gestures. The actual task given the conversants in the experimental protocols—joint recall of a random sequence of letters—met these requirements. Protocols were obtained of three trials each of two pairs of subjects. The protocols were transcribed for verbal and non-verbal behaviors, and the behaviors were noted as possible meta-locutionary acts.

In the next chapter I describe how the behaviors were connected to the acts developed as part of the theory of meta-locutionary acts. These acts, and the conversational models which they affect, explicate the conversation in meta-locutionary terms.

CHAPTER V

RESULTS OF THE PROTOCOL STUDY

In this chapter I focus on the part of the protocol reported in Appendix A. This protocol was the second of the three protocols obtained from these subjects. Recall that the first protocol was intended as a introduction to the problem, and that I expected that in the second protocol the subjects would have a good notion of the task and how to accomplish it.

Modeling the Conversation

Having transcribed the protocol as described in Chapter IV, the illocutions of the acts noted in the transcripts were then developed through interpretation of (1) the specifics of the verbal and non-verbal behaviors associated with the act, (2) the known context of the conversation, and (3) the context of the conversation reasonably imputable to the subjects. In Cohen's (1984) terms, I derived the illocutionary acts as a rational strategy of action, given attributions of the the participants' beliefs, goals, and expectations at the point in the discourse in which the illocutionary acts actually occurred. Note that the apparent subjectiveness of this method is both unavoidable and unobjectionable. It is unavoidable in a qualitative study which seeks to model otherwise unobtainable mental states; post-elicitation of subjects' mental states about comprehension and production are likely to produce confabulations, because the states and acts are generally unconscious ones. The subjectiveness is unobjectionable because there really is no single objective account of the conversation. Although the the conversants are repairing a shared model of the conversation, the sharedness is a subjective quality; they do not really share the actual verbatim states. Therefore, an interpretation of the interaction which plausibly explains the interchange can be taken as both valid and useful. This approach is consistent by recent work in understanding language as an action situated in its context (Suchman, 1987). In this view, the aim of research into language as action is not to produce formal models but to explore the relation of knowledge and action to the particular circumstances in which knowing and acting occur. This approach requires changes in the methodology of research on purposeful linguistic action:

> The first [change] is a fundamental change in perspective, such that the contingence of action on a complex world of objects, artifacts, and other actors, located in space and time, is no longer treated as an extraneous problem with which the individual actor must contend, but rather is seen as the essential resource that makes knowledge possible and gives action its sense. The second change is a renewed commitment to grounding theories of action in empirical evidence: that is, to building generalizations inductively from records of particular, naturally occurring activities, and maintaining the theory's accountability to that evidence. Finally, and perhaps most importantly, this approach assumes that the coherence of action is not adequately explained by either preconceived cognitive schema or institutionalized social norms. Rather, the organization of situated action is an emergent property of moment-by-moment interactions between actions, and between actors and the environment of their action. (Suchman, 1987, p. 179)

Accordingly, based on the hierarchy of illocutionary acts in conversation presented in Figure 2, and its elaboration in Chapter III into a taxonomy, an initial interpretation of the illocution of the conversants' acts was determined. This set of acts and their mapping onto conversational phenomena were refined in successive passes back over the protocol. The final set of acts is listed in Appendix B, Predicate Representations.

Modeling of Meta-Locutionary Acts

From the transcribed and coded protocol, I developed a representation of one of the conversants' model of their conversation. That is, as described in Chapter II, the conversants are viewed as jointly constructing a conversation. Each conversant is checking his or her model of the constructed conversation against the actual evidence of the conversation, namely the utterances themselves in their context. Although the terms may be slightly confusing here, the representation which I developed of A's model is, in effect, a hypothesis as to a set of beliefs which would permit a rational agent to achieve this particular coherent conversation under the contextual circumstances. Thus beginning with some reasonably inferred state, A's model can be updated act-by-act to reflect additions and deletions from the state of the conversation. Figure 10 shows the initial second of the protocol, the initial state of A's model (after B's act Bi1, which is directed to the experimenter and is not further considered here), the changes in A's model resulting from A and B's conversational acts, and the state of A's model after A's act Ai2.[9] The initial state reflects A's immediate domain knowledge and her immediate domain goal. The changes in A's model due to the acts of both A and B represent A's perceptions of the effects on the shared conversational model. Note that a similar set of perceptions, changes, and states can be constructed for B. Both A and B's models, though, should be viewed as representing what A and B respectively believe the state of the conversation to be. Thus from an initial state of (i) A's not knowing the first letter of the sequence and (ii) A wanting B to inform her of the first letter, the sequence of acts Bi2, Ai1, Ai2 results in the state of A's model being that (i) A does not know the first letter of the sequence, (ii) A wants B to inform her of the first letter, (iii) it is A's turn in the conversation, and (iv) A has informed B that A does not know the first letter.

[9]In Figure 10 and in the discussion, these notational conventions are followed: Indices: T time, Av A's verbal, Bv B's verbal, Ap A's physical, Bp B's physical, Ai A's illocution, Bi B's illocution; Body: h head, e eyes, a arm, ah hand, r right, l left, f finger, m mouth; Directions: U up, D down, L left, R right; Referents: A conversant, B conversant, O object, E experimenter; Actions: W forehead-wrinkles, C forehead-clear, O open-mouth, S close-mouth; Notation: () previous state, ... duration.

Using this technique, the state of A's model of the conversation was determined after each act identified in the transcribed segment of the protocol. This account of A's model is set out in Appendix C.

```
T   00.0    00.2    00.4    00.6    00.8    01.0...
    Av                   blank..................
    Ap (turn'g-hUBeB)hDB...
    Ai                Ai1 Ai2
    Bv 'kay...................
    Bp (hAeA,rlhU)    rlhD,together,leaning-back
    Bi Bi1            Bi2


Bi1:   {indicating that conversation can start}
      = not(know(A,<first letter>))
      = wants(A,inform(B,A,<first letter>))
Bi2:  give-turn(B,A)
      + accedes(B,(turn(A))
Ai1:   acknowledge-turn(A)
      - accedes(B,(turn(A))
      + turn(A)
Ai2:   inform(A,B,{?X = doesn't have first letter})
      + informed(A,B,not(know(A,<first letter>)))


not(know(A,<first letter>))
wants(A,inform(B,A,<first letter>))
turn(A)
informed(A,B,not(know(A,<first letter>)))
```

FIGURE 10. Illocutionary interpretation of the protocol. The figure contains the first second of the protocol, the initial state of A's model (after B's act Bi1, which is directed to the experimenter and is not further considered here), the changes in A's model resulting from A and B's conversational acts, and the state of A's model after A's act Ai2. '=' denotes a predicate initially part of the state, '+' denotes addition of a predicate to the state, and '-' denotes removal of a predicate from the state.

Conversational Operators

From the illocutionary acts and the state-models representations of the conversational structure, I initiated development of a set of operators for the acts, as delineated in Appendix D. The representation used here and in Appendix D is a simplification for purposes of clarity of the more complex form of rule actually used in the simulation discussed in Chapter VI. An example of one of the conversational operators is presented in Figure 11. The left-hand side "IF" part of the operator represents a set of felicity conditions for execution of the operator. The right-hand "THEN" clause of the operator identifies the act (or acts) to be taken if the "IF" conditions are satisfied. These acts constitute the conversant's contribution to the structure of the conversation. The changes in the conversant's account of the mutual model of the conversation are noted in the "EFFECTS" part of the operator.

```
IF        not(mutually-known(me,Other,wants(me,Act))
          request(me,Other,Act)
          turn(me)
THEN      repeat(request(me,Other,Act))
EFFECTS   - not(mutually-known(me,Other,wants(me,Act))
          + mutually-known(me,Other,wants(me,Act))
```

FIGURE 11. Operator repeat-act-1. The left-hand-side clauses are matched against the conversants model. If all clauses are true and the operator is executed, the acts "repeat" and "give-turn" are performed, with their variables instantiated as matched in the "IF" clauses. The conversant's model is modified by the accordingly instantiated "EFFECTS" clauses.

Thus given the state of A's model of the conversation just after act Bi6, the "IF" part of the operator matches clauses 7 and 2 of the model, plus Act Ai4 of the conversation. This instantiates the operator as shown in Act Ai5. The instantiated "EFFECTS" clauses are then applied to the state, resulting in the new state following the act. The details of process are set out in Figure 12. Examples of meta-locutionary conversational operators developed from analysis of the protocols are presented in Appendix D.

```
A's state after act Bi6:
    (1)  wants(A,inform(B,A,<first letter>))
    (2)  turn(A)
    (3)  informed(A,B,not(know(A,<first letter>)))
    (4)  mutually-known(wants(B,turn(B)))
    (5)  wants(B,clarify(A,?X)
    (6)  mutually-known(not(know(A,<first letter>)))
    (7)  not(mutually-known(wants(A,inform(B,A,<first letter>))))

A's act Ai5:
    repeat(A,request(A,B,inform(B,A,{?Y = first letter})))
    give-turn(A,B)

Effects on A's model:
    - not(mutually-known(wants(A,inform(B,A,<first letter>))))
    - mutually-known(wants(B,turn(B)))
    - turn(A)
    + mutually-known(wants(A,inform(B,A,<first letter>)))
    + turn(B)

A's State after act Ai5:
    (1)  wants(A,inform(B,A,<first letter>))
    (3)  informed(A,B,not(know(A,<first letter>)))
    (5)  wants(B,clarify(A,?X)
    (6)  mutually-known(not(know(A,<first letter>)))
    (8)  mutually-known(wants(A,inform(B,A,<first letter>)))
    (9)  turn(B).
```

FIGURE 12. Application of operator repeat-act-1 after act Bi6. The enumerated facts represent the state of A's active memory. At the beginning of the figure, A's state of understanding of the conversation is portrayed immediately after B takes act Bi6. Based on this state, then, A takes the acts which make up act Ai5. The effects of these acts on A's model of the conversation are then set out. The figure concludes with A's new model of the conversation, after application of the effects of act Ai5. Note that the changes in conversants' turns are effected through a simultaneous application of a give-turn operator

From these operators conversational plans can be developed. Such plans are not intended to describe multiple turns but rather are intended to show how different operators can be combined through a planning process to achieve complex acts within a turn. For example, if the conditions for application of repeat-act-1 were present except for turn(me) then an operator such as T1 might be applied to facilitate repeat-act-1. I want to stress here that the use of operators which can be embodied in rules does not mean that I have proposed a planning system. In the set of operators which were developed from the protocol study and were adapted into the simulation, none relied on backchaining or chained through a single cycle. Rather, the operators were constructed to model (1) situated responses to local context, (2) goal-directed inference reflecting intentionality, and (3) understanding. Conversations resulting from application of the operators as written achieve their interactive behavior entirely through recognition of situations and then consequently posting goals. Suchman (1987) has observed that real conversations are not planned top-down, although later analysis can usually parse them into plan-like structures:

> While the organization ... of any interaction can be analyzed post hoc into a hierarchical structure of topics and subtopics, or routines and subroutines, the coherence that the structure represents is actually achieved moment by moment, as a local, collaborative, sequential accomplishment. This stands in marked contrast to the assumptions of students of discourse to the effect that the actual enactment of interaction is the behavioral realization of a plan. Instead, every instance of coherent interaction is an essentially local production, accomplished collaboratively in real time. (Suchman, 1987, p. 94)

Accordingly, the operators proposed here are to be interpreted as constituting local behaviors which, in the aggregate, produce coherent linguistic action. In the implementation of the model as a rule-based system, as discussed in Chapter VI, the felicity of every operator is assessed on each cycle. Of course conversations have structure; they are intentional processes. In the model proposed here, intentionality is represented by the posting of goals in active memory. Whether a goal is subsequently attained is not a function of the operator that posted the goal. Rather, achievement or abandonment of a goal is a consequence of matching and executing of later instantiation of operators in response to the local conditions then existing. A person may enter into a conversation intending to ask after the health of the other party's spouse but never attain that goal because other more urgent matters capture the conversational focus. The person is not stuck with a stack-based planning model which prevents the person from concluding the conversation without mentioning the other's spouse. Such goals are simply abandoned in the face of later local situations. The meta-locutionary model, then, despite relationships among operators, has more of an opportunistic control structure than that of traditional planning systems. The overall structure of a conversation certainly exists in the sense that a given initial—or subsequent, for that matter—state of a conversant's memory plus his set of operators create, in effect, an expectation as to the conversation's path. Yet the coherence of the interaction does not depend on that specific path being followed. Each change in the state as a result of interaction creates a new implicit path for the conversation (which could be the same as the old path if the interaction follows expectations). It is also important to note that the conversants' acts noted in the protocol are complex: they are effected through multiple simultaneous acts at different conversational levels. For example, A's act Ai5 shown in Figure 12 consists of acts at both the repair and turn-taking levels.

Application of the Model

The modeling techniques described above permitted development of plausible explanations of a number of interesting aspects of the conversation. Here I analyze a portion of the protocol in terms of meta-locutionary acts involving reference, turn-taking, and repair. The relevant section of the transcript is presented in Figure 13. The analysis suggests that even in such an apparently simple domain as the joint-recall task, the conversants encounter coherence difficulties which require relatively complex meta-locutionary interaction to resolve. Further, the conversants' speech acts demonstrate an information-systemic rationale for the use of indirect speech acts; this arises from the conversants' mutual responsibility for maintenance of the conversational belief structure.

```
T  00.0   00.2   00.4   00.6   00.8   01.0   01.2   01.4   01.6   01.8...
==========================================================================
Av                   blank................................
Ap (turning-hUBeB) hDB...
Ai                 Ai1Ai2
--------------------------------------------------------------------------
Bv 'kay....................
Bp (hAeA,rlhU)   rlhD,together,leaning-back    mO.......
Bi Bi1           Bi2                           Bi4

T  02.0   02.2   02.4   02.6   02.8   03.0   03.2   03.4   03.6   03.8...
==========================================================================
Av                 is...the.....first...one....so....what...was.the.first
Ap        eBlink
Ai        Ai3        Ai4                                ....Ai5
--------------------------------------------------------------------------
Bv                                               "O"...........
Bp                                mO
Bi                                Bi5             Bi6

T  04.0   04.2   04.4   04.6   04.8   05.0   05.2   05.4   05.6   05.8...
==========================================================================
Av ....one........                                  OK..........
Ap                                                       eBlink
Ai                                                  Ai6  Ai6a
--------------------------------------------------------------------------
Bv                             "O"......                       an'.I.....
Bp                  eBlink
Bi                  Bi7        Bi8                             Bi10

T  06.0   06.2   06.4   06.6   06.8   07.0   07.2   07.4   07.6   07.8...
==========================================================================
Av            "I".................           "S"..............    "U"....
Ap         hNod                        hNod/eBlink            hNod/eBlink
Ai         Ai7Ai8                      Ai9   Ai10                  Ai11
--------------------------------------------------------------------------
Bv ...                                                            "U"..
Bp                                hNod,hNod/eBlink.                hNod
Bi                                Bi10 Bi11                        Bi12
```

FIGURE 13. Partial transcript of experimental protocol. This conversation is the beginning of the second trial for subjects A and B.

Reference/Information Acts

One level of the taxonomy of meta-locutionary acts presented in Chapter III was the reference/information level. I begin the analysis of the protocol section by looking at various acts at this level.

Request

A's act Ai2, about 0.5 seconds on the transcript timeline in Figure 13, is an act of requesting that B provide information about the first symbol in A's sequence, a blank. This interpretation is plausible in light of the following factors: (1) We believe that A understands the experimental task of joint recall because she competently performed the task for the first trial. Accordingly, she will have as a goal the act of confirming the entire sequence as mutual. (2) Moreover, we believe that A knows that her first symbol is a blank (because she says so). This leads to the reasonable assumption that A has developed a subgoal of obtaining from B the letter which corresponds to the blank. Therefore, A's act is probably something like request(A,B,{first letter is "blank"}).[10] After act Ai2, A's active-memory model of the conversation will then be something like[11]

```
not(know(A,<first letter>))
goal(A,assert(B,A,<first letter>))
turn(B)
mutually-known(A,B,not(know(A,<first letter>))).
```

[10]Actually, Ai2 is somewhat more complex in that it also has a turn-taking component. The turn-taking aspects of the act are discussed later in this section.

[11]The notation used in these examples is more abstract than that used in the simulation. Conversants no doubt keep track of additional information about these states, including temporal information. For purposes of understandability, the memorial predicates used here are simplified to make clear their most relevant features.

Assert

At Bi6, B's act—in response to A's act Ai4—is assert(B, A, first letter is "O"). This interpretation is a plausible one in light of three factors. First, B apparently understands the experimental task and can be construed as having the goal that both conversants have mutual knowledge of the first letter of the sequence. Second, A's request (combining Ai2 and Ai3) was made as part of the conversation, so the conversants have directly copresent knowledge of the request; it would be plausible in the conversational context, then, for B to have responding to A's request as a specific goal. Third, B's first letter is, in fact, "O." His act at Bi6, then, can be seen as an assertion which provides information about a conversational referent, namely the first letter of the sequence. B's active-memory model of the conversation is now probably something like

asserted(B,{first letter is O})
turn(B).

Acknowledge

At Bi8, B has (again) asserted that the first letter was "O." At A's next act, Ai6, she confirms that she understood that the first letter was "O." Her act is probably something like acknowledge(A,{first letter is O}), confirm-mutual(A,B,{first letter is O}). Note that the second part of the act is at the mutual-knowledge-repair level, for reasons discussed with respect to repair below.

Another example of an acknowledge act is at Ai7. B has, at Bi10, just said "an I ..." Thus as A knows that the next letter is in fact "I" she acknowledges B's assertion with her act Ai7, which might be coded as acknowledge(A,assert(B,{next letter is I})).[12]

[12]It turns out that B's act Bi10 could not have been assert(B,<next letter is I>) because he did not know the next letter was "I." I examine the consequences of this in the

Turn-Taking Acts

The protocol contains examples of situations both where turn-taking occurred and where it might have occurred. At Ai3 A could have given a turn but does not do so. The context for this act is that B has invited A to speak, A says "Blank," B indicates that he wants to take a turn, and then A blinks. At this point A does not give the turn to B, though, since she goes on to extend her verbal utterance. In the meta-locutionary interpretation of the protocol, B's act Bi4 constitutes a control act for the conversation, and A's subsequent act Ai3 is an acknowledgment of this.[13] Note though, that this acknowledgment, as lexicalized, is more of a place-holder than a full-blown expression of comprehension. This act is, in a sense, a vestigial one: it marks a place in the conversation where A would have interposed an indication of miscomprehension had she failed to understand B's act. As A in fact did understand the act, no indication of miscomprehension was needed. It aids the conversational process, though, for A to mark this place in the conversation because, absent such a marker, B may be forced to wait an indefinite period for the miscomprehension indicator before he continues. Accordingly, A's act at Ai3 can be seen either as an acknowledgment of B's act at Bi4 or as holding her turn—because B didn't respond for almost a second after A said "Blank."

---

discussion of repair acts below. Nevertheless, A's act Ai7 is still valid, because it depends on *her* interpretation of what B meant at Bi10.

[13]The analysis here of A's blink at Ai3 as an act of acknowledgment does not imply that blinks can be uniformly interpreted as having this meaning. First, the meaning of action is provided by its context; a blink under different circumstances will communicate different information to the other conversant—even "I've got some dust in my eye." Second, actions may be involuntary concommitants of communication. See the discussion of intention and action in Chapter III and the discussion of lexicalization in Chapter VI.

Give-turn

A's act at Ai2, where she utters "Blank," can be viewed as a complex act. It is not only a request, as discussed above, but also the giving of a turn. The fact that A expects B to take the turn after Ai2 is apparent from A's subsequent pause between 01.6 and 02.5 seconds on the protocol timeline. A's complete act at Ai2, then, is probably something like request(A,B,{first letter is "blank"}), give-turn(A,B). This is a logical consequence of the circumstance that A is making a request of B.

Similarly, A's act at Ai6a can also be viewed as something like give-turn(A,B). Here A has just acknowledged at Ai6 that she understands B's assertion that the next letter is "O." In the absence of a turn-taking or turn-giving act, the control of the conversation would appear to rest with A. B's subsequent picking up of the conversation at that point is evidence that the turn has been given to him.

Acts Repairing the Mutual Model

As was seen to be the case with turn-taking, there were contexts where an act occurred and other contexts where an act might have occurred but did not. Similar circumstances occur with respect to acts which repair the conversants' mutual model of their interaction. The experimental protocol is replete with occasions where one or both of the conversants is in a position to detect divergence of their models of the conversation. In some cases, repair acts are undertaken; in others, repair is apparently deemed not necessary. The following discussions examines examples of both cases.

Clarification Repairs Made

A's acts at Ai4 and Ai5 can both be viewed most plausibly as acts of repair of the mutual model of the conversation. At Ai2, A has—she believes—made a request for B to assert the first letter of the sequence because she has a blank. She waits, she acknowledges that B seems to want a turn, and yet B doesn't say anything. At Ai4, then, A elaborates her utterance to make explicit the informational content of her act at Ai2.

At about 3.3 seconds into the conversation, at act Bi6, B asserts that the first letter of the sequence is "O." A, however, goes on at act Ai5 to request explicitly that B tell her the first letter of the sequence. Why, then, does A take act Ai5?

Conversant A naturally enough does not have access to either B's actual intentions in saying "O" or to any genuinely objective account of the shared conversational structure. Rather, she depends on her own interpretative faculties to place B's utterance coherently into their conversation. In this case, A's action can be accounted for by realizing that she interpreted B's utterance "O" as "Oh"—that is, acknowledging A's statement at Ai4 but not telling her what the letter actually is. The effect of this misunderstanding can be traced through A's model of the conversation. Before B's act Bi6, A's model is

```
wants(A,inform(B,A,<first letter>))
turn(A)
asserted(A,B,not(know(A,<first letter>)))
mutually-known(wants(B,turn(B)))
wants(B,clarify(A,?X)
mutually-known(not(know(A,<first letter>))).
```

At Bi6, B's act (from B's point of view) is assert(B, A, {first letter is "O"}), but A hears "Oh." and thinks that B's act is mutually-known(A,B,not(know(A,<first letter>))), but not acknowledge(B,request(A,B,assert(B,A,{first letter})))). As a consequence, A updates her model of the conversation as follows:

```
- not(know(A,<first letter>))
+ mutually-known(not(know(A,<first letter>)))
+ not(mutually-known(wants(A,assert(B,A,<first letter>)))).
```

Applying these changes, A's model is now:

```
wants(A,assert(B,A,<first letter>))
turn(A)
```

```
asserted(A,B,not(know(A,<first letter>)))
mutually-known(wants(B,turn(B)))
wants(B,clarify(A,?X)
mutually-known(not(know(A,<first letter>)))
not(mutually-known(wants(A,assert(B,A,<first letter>)))).
```

Note that A and B's models have diverged significantly at this point, since B's model of the conversation, as discussed above with respect to assert acts, presumably contains something like

```
asserted(B, A,{first letter is "O"})
mutually-known(A,B,{first letter is "O"}).
```

In A's version of the conversation, though, the state is that A has told B that her first letter is blank, B has indicated that he has understood this, but B has not told her what the letter was. A thus concludes that B must not have interpreted her utterance at Ai4 as a request. This leads her to then repeat her request in act Ai5. When B in turn repeats "O," A presumably realizes their misunderstanding and updates her model accordingly.

It could be argued that B's long pause (between 01.5 and 02.5 seconds of the transcript) before A again begins speaking represents his non-comprehension of A's act at Ai2. This interpretation, however, is implausible in light of the express negotiation of interaction patterns which had just taken place a few minutes before in the previous "training" protocol. It is consistent with the negotiated forms of interaction from the first protocol for B to understand that A had a blank for her first letter. If anything, B's pause should be ascribed to an expectation that A was going to continue recounting her sequence.

It may also be true that physical constraints on cognition are determining some aspects of A's utterance at Ai5. If she hears B say "O" while she is producing the utterance, she may simply finish her utterance. In either case, though, B hears A make a clarification of her original request.

Implications for Computational Views of Indirect Speech Acts

The modeling and explication of this exchange also has some implications for the theory of indirect speech acts. Unlike indirect acts resulting from politeness (as discussed in Chapter I), here we see evidence of indirection arising from coupling an interest in for efficiency with the creation of the conversation as a shared structure. A's act at Ai2, where she opens the verbal part of the conversation with "Blank," is indirect. Her direct act is, on its face, an assertion, but she clearly intends the utterance to function as a request. B certainly interprets it as a request. Thus if the underlying logical structure of A's act was really request(A,B,inform(B,A, ?Y = first letter))), why didn't she just make a direct request or question along the lines of "What's the first letter?"

The reason for A's indirection comes from the meta-character of A's discourse interaction. She is engaged in the process of building with B a mutual model of the context. For both A and B to have mutual knowledge of the true character of the sequence, A needs to tell B that she has blank for her first letter. But, if this is the case, why then did A not say something like "I have a blank so what's the first letter?" Here the reason appears to be efficiency. It is more efficient just to assert "I have a blank" because this utterance also functions as an indirect speech act constituting a request. A's striving for efficiency is apparent from the form of her actual locution, which is simply "Blank." When A then misapprehends that B has perceived only her direct act and has failed to perceive the indirect act, she then produces the request as a direct act: "... so what is the first letter?"

This analysis thus suggests that some indirect speech acts may result from meta-locutionary attempts to maintain the shared model of the conversation while interacting efficiently. This is a systematic, as opposed to a socially conventional, motivation for indirect speech acts. That is, while some acts appear to establish social relationships or conform to social expectations of politeness, the indirect act taken by A here appears to arise directly from the nature of language as a medium for interaction.

Clarification Repairs Not Made

In the preceding discussion, I suggested that at Bi6 B intends to tell A that the first letter is "O" but A understands B as meaning "Oh" and simply confirming that he understood her. Having just gone through this miscommunication and minor repair, B is again coïncidentally misunderstood by A in his act Bi10. Interestingly, the misunderstanding is precisely the converse of what they had just experienced. At Ai6a gives the turn back to B. B then says "an' I ..." and A immediately confirms at Ai7 and Ai8 that the next letter is indeed "I." A's further recitation of the succeeding letters is evidence that she believes that their understanding is mutual that the second letter is "I." At Bi10, then, it is probable that A interprets B's act as something like inform(B,A,{next letter is "I"}). As a result, A's model of the conversation is plausibly in this state:

mutually-known(A,B,{first letter is O})
turn(B)
mutually-known(A,B,{second letter is I}).

However, A's interpretation, which then leads to her acts at Ai7 and Ai8, is mistaken. Obviously, she does not have direct access to B's sequence and therefore did not know that B's second letter was a blank! In other words, if we were following B's model instead of A's, B's act at Bi10 is actually the start of something like inform(B,A,{my next letter is a blank}. Lexicalized, the full locution of B's aborted utterance would probably have been "and I ... have a blank." Through coïncidence, A's second letter turned out to have been "I," so she heard B's utterance not as a word but as a letter. Note that this is virtually the exact converse of her misunderstanding about "O."

In this case, though, no repair utterances ensue. This appears to be the result of fairly rapid inference on B's part. A stretches out her locution confirming "I" and after 0.5 second B nods twice rapidly and blinks (perhaps in astonishment). B does not need to repair the conversation because at that point B knows that the knowledge that "I" is the second letter is mutual. At this point, then, the conversants have different models of what they think the discourse is, but only B knows this. He's satisfied to continue confirming the recitation of the subsequent letters because the future path of the conversation is not likely to be adversely affected. Eventually the conversants complete their joint recall of the sequence without A ever learning that B's second symbol was a blank.

Consequences of Repairs

The attenuated acknowledgment at Ai3 is quite a contrast to the explicit verbal acknowledgment given by A after B's act Bi8. When at Bi8 B repeats "O," A responds at Ai6 with "OK." In both cases A is effectively confirming the mutuality of the shared model of the conversation. Why, then, are the acts she takes so different? The answer, as suggested in the model set out in Appendix C, is that in Ai6 A was coping with the consequences of the preceding misunderstanding. If this had been a response to B's first utterance "O" at Bi6, a normally vestigial response might well have been appropriate, since in this case there would have been no miscomprehension. The actual circumstances at Ai6, though, were that miscomprehension had in fact been communicated by A, so a place-marker response could be seen as ambiguous; could be expecting additional clarification. Thus an explicit affirmative response was thus called for.

These differences can be seen in the particulars of the acts which A takes at Ai3 and Ai6. At Ai3, A's act is simply assert(A, B, comprehend(A, take-turn(B))). At Ai6, her act is necessarily more complex: acknowledge(A, {first letter is O}) and confirm-mutual(A, B, first letter is O). It is the addition of the confirm-mutual act which results in the explicit acknowledgment.

The Computational Utility of Meta-Locutionary Acts

In looking at examples of various meta-locutionary acts at different conversational levels, I have tried to point out how explication of the interaction in terms of these acts provides a plausible, rational basis for reconstruction of the conversants' linguistic behaviors. In considering turn-taking, for example, the analysis suggests that meta-locutionary acts at the turn-taking level can be applied to a representation of the conversational context to produce an adequate explanation of control. These qualities might be extended to actual control of interaction through appropriate implementation of the operators. In Chapter VI, then, I present a partial implementation of the model in order to show that the theory is sufficient to produce behaviors that, under simulated conditions similar to those encountered and created by the actual conversants, reasonably replicate the kind of interaction observed in the protocol.

CHAPTER VI

THE SIMULATION STUDY

To test whether meta-locutionary operators would plausibly explain or reproduce the behaviors observed in the protocols, a forward-chaining rule-based simulation was built following the computational model developed through protocol analysis. The simulation was a partial implementation of the theory as a rule-based system. This work was done in two parts: first, the creation of a meta-interpreter in Prolog in which to write and execute rules; and second the development and running of rules which represent the meta-locutionary operators. The goals of the simulation were to determine if the theory and model actually produced plausible control in interaction and to serve as an experimental environment for meta-locutionary operators. The system has some similarities with that written by Power (1979), who modeled a plan-based conversation between two robots named John and Mary. Here, for example, is the beginning of an exchange:[14]

1. JOHN: MARY. (call by name)
2. MARY: YES. (acknowledgment)
3. JOHN: I WANT TO SUGGEST A GOAL. (ANNOUNCE 1)
4. MARY: GO AHEAD. (ANNOUNCE 2)
5. JOHN: WILL YOU HELP ME GET IN? (AGREEGOAL 1)
6. MARY: BY ALL MEANS. (AGREEGOAL 2) ....

[14]In this transcript Power added parenthetical information about which conversational procedure was being used. This information was not used by the agents in his system and is provided only for the convenience of the reader.

7. JOHN: SHALL WE MAKE A PLAN? (ANNOUNCE 1)

8. MARY: JOHN. (call by name)

9. JOHN: YES. (acknowledgment)

10. MARY: MAY I ASK YOU SOMETHING? (ANNOUNCE 1)

11. JOHN: GO AHEAD. (ANNOUNCE 2)

12. MARY: ARE YOU IN? (ASK 1)

13. JOHN: NO. (ASK 2) (Power, 1979, p. 116)

The simulation which is described in this chapter is similar to that by Power, except that the meta-locutionary simulation (1) goes beyond representation of illocution to address control issues, (2) uses situated action in place of top-down planning; and (3) models observed instead of idealized interaction.

Representation

Both development of the rule-based system and creation of rules embodying the meta-locutionary operators depended on a representational scheme for conversational models of conversants. The representation for both the mental states and the operators was necessarily more complex than that for the more abstract states and operators used for the protocol analysis. Natural language was not used directly as the representation of either states or operators. With respect to the internal representation, there is no reason to believe that language is a plausible representation for mental states. Stucky (1988) pointed out that utterance structures, even translated into predicate logic, are unlikely to be the key to adequate representations of successfully understood utterances. Such successful states must be formed in such a way that they can tie indexicals in an utterance to features of the context. That is, the situated character of language action must be accounted for in the representation. With respect to the operators and their output,

> A difficulty in writing programs of this kind is that to understand the point of a remark one must normally begin by understanding its literal meaning. Assuming that the programmer has nothing to say about the latter, he must either: (a) run the conversation in the program's internal semantic notation, (b) translate between this notation and English by unprincipled short cuts. (Power, 1979, p.109)

Thus the meta-locutionary simulation expresses its actions in its internal representation; to do otherwise would be misleading, because the model does not really address issues of lexicalization at all.[15] There would be an appearance of linguistic skill not actually found in the model. Interestingly, it would have been possible to express the output acts of the simulation in natural-language form, perhaps making the exchanges more immediately understandable.

In the rule-based simulation, the conversants' mental states were represented as aggregations of currently-active individual states. These states were either beliefs or acts. Of course, the mental representation of an act in some sense is also a belief. However, the propositional content of acts seems distinct from other statements about the world. Acts have an inherent temporal aspect; they occur rather than exist. An act of clarification has exists while it is happening, but then has no existence independent from its recollection. It may have been true that an act took place, but it is no longer a part of the world except for its effects, including its own recollection. In contrast, beliefs about the world may have a temporal element, but they generally also have a continued existence that is dependent on but not derived from memory. For example, the belief that someone is confused has validity beyond the moment of recognition of the state; such a belief has existence over some period of time.

Accordingly, the computational model represents acts and beliefs as follows:

[15]Actually, Power (1979) took the second approach so that observers could more easily follow the conversational flow. He built a simple expectation-based parser which looked for key words.

```
state (act (Actor, <act>(<act args>), Belief_state), Cycle_added,
        Cycle_deleted)

state (believe (Actor,<belief>(<belief args>), Belief_state), Cycle_added,
        Cycle_deleted)
```

This notation is similar to the Prolog-style syntax used in representing the protocol. Lower-case names represent predicate labels, capitalized names are variables, and <bracketed> items represent either predicate labels for acts or beliefs (give_turn, next_letter), or values for arguments to the predicates.

The Belief_state variable contains a value expressing the truth value of the belief or act. This approach has a number of advantages. It permits the direct and explicit representation of belief in falsity rather than relying on non-existence to stand for falsity. It also encodes intentionality. The full set of values for belief state are true, false, mutually_known_true, mutually_known_false, goal_true, goal_false, goal_mutually_known_true, and goal_mutually_known_false.[16]

The Actor variable is filled by the name of one of the conversants. The cycle values indicate when a state entered and left memory. They do not indicate that a fact became true or false; truth values are a function of the Belief_state variable. A negative value for Cycle_deleted indicates that the state is still in active memory.

Using this notation, then, the state

```
state ( act ( angela , give_turn ( phil ), goal_true ), 5, 8)
```

[16]This list of belief values does not contain explicit provisions for dealing with uncertainty. In practice, a conversant's noting in memory that some fact about the conversation is true can serve as a hypothesis which is later confirmed as mutually_known_true. Adding additional levels of (real) uncertainty would have been difficult to handle gracefully in the rule-based system without adding near-duplicate rules to handle alternative truth values. However, a reasonable use for qualitative truth values might involve meta-control in conflict resolution, where choices of what action to take could depend—either way—on the certainty of the conversant's knowledge.

means that the conversant in whose memory this state is placed had as an active fact from cycles 5 to 8 a belief that Angela wants to give the turn to Phil. This state alone does not indicate why the fact was removed from active memory. It may have been that the fact was simply no longer relevant through the action of an operator. It may have been that the fact, which expressed an intention, was not needed after cycle 8 because the goal was no longer being sought. In the current implementation of the system, the cycles are used by the matcher and conflict resolver, but are not available to the operators. The implications of this limitation are discussed below.

A conversant's memory, then, consists of a bag of such states, where the cycles indicate what is currently believed and what was believed before. Nothing in the system automatically prevents the insertion of conflicting states. It is left to the operators to assure that the states achieved are rational ones.

The sequence-recall-task domain also presented issues for knowledge representation. It was apparent from the protocols that although the conversants could recite the entire list of letters, they dealt locally with confirmation and communication. Typically, conversants would go through a short part of the sequence of letters, and then explain that these letters were followed by N blanks. In effect, they created subsequence substructures from the sequence as whole using the apparent features of the otherwise random sequences. This behavior is consistent with that observed in psychological studies of the ability to recall or to predict sequences. People tend to induce pattern descriptions from the sequences and rely on notions of *same* and *next* alphabetic characters, iteration of subpatterns, and hierarchic phrase structure (Simon, 1972). The maximal length of the subsequences in the model was set to either (1) the extent to which the symbols were of the same sort (i.e., letter or blank), or (2) a chunking constant representing the capacity of short-term memory. In this study, the chunking constant was set to five, which appears to be a psychologically plausible value for sequence recall (Simon, 1974). The domain-related functions which created the subsequences are listed in Appendix J. As implemented, subsequences and letters have indexes back into their super-sequence.

The operators are represented in standard rule form, as shown in Figure 14. The "if", "not", and "test" parts of the rule represent a differentiation of functions grouped together in the simpler "IF" part of the conversational operators discussed in Chapter V. If the "if" clauses are true, the "not" clauses cannot be proved, and the "test" predicates are true, then the operator is instantiated because its felicity conditions obtain. If the operator is executed, the "action" acts are posted to the memories of both conversants and the "effects" states are added or deleted as indicated from the memory of the conversant executing the operator. The attributes contain name and level information useful to readers and to the conflict resolver; the levels are intended to correspond to the levels of illocutionary acts presented in Figure 2.

```
if ([ <state1>, <state2>, ... ]),
not ([ <state3>, <state4>, ... ]),
test ([ <predicate1>, <predicate2>, ... ]),
action ([ <act1>, <act2>, ... ]),
effects ([ add (<state5>), del (<state6>), ... ]),
attributes ([ name (<operator_name>), level ( <illocutionary_level>)] ).
```

FIGURE 14. Form of rule representing meta-locutionary operator. The "if" clauses are matched against the conversant's active memory. The "not" clauses succeed if they cannot be matched against memory. The "test" clauses are predicates, written in Prolog, which must succeed if the rule is to be fired. If all clauses are true and the rule is fired, the acts contained as "action" clauses are added to the memories of both conversants, with their variables instantiated as matched in the "if" clauses. The conversant's model is modified by the accordingly instantiated "effects" clauses.

The Rule-Based System

I now turn to a description of the forward-chaining rule-based system with which the computational model was tested. I first discuss the specifications for the system generally, and then report on how the specifications were met with the system as implemented.

Specifications

In the abstract, the rule-based system had to meet certain criteria. First, it had to be capable of reasoning from the predicate representations of its facts. Second, it had to have independent knowledge bases for the different conversants while allowing for communicative action. Third, it had to be capable of simulating simultaneous action by the conversants. And fourth, it had to allow customizable functions for conflict-resolution. To be able to reason from the predicate representation of the facts, the system was built on top of Prolog. It turns out that as implemented, virtually all of the domain and meta knowledge was contained directly in the operators rather than in a separate planning or reasoning subsystem.

Having independent knowledge bases was critical to the simulation. The concept of conversation as a jointly constructed entity meant that each conversant maintains their own model of the conversation while believing the model to be shared one. To the extent that divergences in the models are detected by the conversants, repair interaction should occur. This means that the knowledge bases had to be set up so that operators could be written generally enough to apply to both conversants yet avoid matching on memorial structures of the other conversant. At the same time, a path between the agents has to exist in order for the acts to be communicated. In the operator representation, the actions are posted as acts in both conversants' knowledge bases.

Simultaneous action has three aspects: inter-agent, intra-agent, and mental-process. Inter-agent simultaneity means having both conversants take their actions at the same mythical time as the other. A normal cycle-by-cycle alternate execution of the conversant's operators would not have permitted simulation of simultaneous action. That is, the model agents must be able to take action on every cycle irrespective of the implemented order of their actual execution. This can be achieved in a specialized rule-based system by designing the cycling functions to take account of the two agents and their independent knowledge bases. In contrast, Power's (1979) agents, named John and Mary, were not capable of simultaneous action. They were separate programs which were explicitly given sequential turns.

Intra-agent simultaneity means allowing the agents to execute more than one instantiated rule per cycle, unlike, for example, YAPS (Allen, 1983) or OPS5 (1981). The hierarchical nature of illocutionary acts suggested by the feedback-based control of natural conversation implies that acts on the various levels are occurring at the same time. Cohen and Levesque (1982) observed that a single utterance may contain multiple acts. This requires special handling of the conflict set. The agents written by Power (1979) did not have this problem because they were plan-based rather than rule-based.

By mental-process simultaneity I mean that the system has to recognize that understanding, reasoning, and action all have temporal extent, and all have to be able to occur both sequentially and simultaneously. That is, while an agent is taking one action, it may also be recognizing an action by the other agent and reasoning about states and goals. Power (1979) did not address this issue.

Finally, the control knowledge needed for conflict resolution has to be domain specific. Aside from the need for executing multiple instantiations, the conflict resolver has to be able to reason about consistent and conflicting acts and effects, and the relationships among different classes of acts. Moreover, conflict resolution may need to be tuned as more experience is gained with the simulation or different hypotheses are tested with respect to reasoning about actions. For example, if there are conflicting effects among acts at different conversational levels, should the system execute the lowest-level operator or the highest-level? Or perhaps the system should use some other, situation-based meta-rules (see Davis, 1980). Although the system is not able to handle meta-rules, it does rely on customizable conflict resolution in the spirit of McDermott and Forgy (1978) and ORBS (Fickas & Novick, 1985). The basic conflict resolution strategy used in the simulation was (1) reduce the conflict set to avoid multiple instantiations of operators at the same illocutionary level; (2) find the lowest-level instantiation, and prefer it; and (3) eliminate any instantiations whose acts and effects conflict with those of the preferred instantiation. The specific functions which handle conflict resolution can be found under the "select_ops" clauses in Appendix G.

Implementation

The rule-based system in which the simulations was conducted was written in SICStus Prolog (Swedish Institute of Computer Science, 1988), running on a VAX 11-750 computer. SICStus Prolog is similar in syntax and essential semantics to DECsystem-10 PROLOG and Quintus Prolog. A listing of the system is presented in Appendix G (although utility clauses commonly used in Prolog programs are omitted).

Basically, the system consists of a set of conventions about representation of states, a rule-matcher, a conflict resolution function, a driver for cyclic action, some I/O functions, and some development tools. The conventions for representation are exactly as discussed above. They had been prototyped as objects in KEE 3.0 and then unfolded into predicate representations for purposes of clarity. The separation of the knowledge bases for the two conversants (named Adam and Barney, representing subjects A and B) was implemented through the use of the names of the conversants as a key into Prolog's internal database. Facts in memory are considered active if they were added before the current cycle and either (1) never deleted or (2) deleted after the current cycle.

The rule matcher takes a straightforward brute-force approach rather than using a rete-style approach. Given the current number of rules (fewer than 40) and the number of facts in the knowledge-base (fewer than 1000 after 20 cycles), the matcher's performance is acceptable. It relies on Prolog's internal unification functions. Note that the operators' "not" clauses refer to the inability to match facts in the memory rather than explicit falsity. This means that the nots-matcher will find explicitly false states in active memory but will also find any non-existent state. This approach is much more efficient than the alternative in representing knowledge in memory. The tests are simply evaluated in Prolog directly, with the variables in the test predicates instantiated from the "if" and "not" clauses. The test predicates can also be used to generate values used in the right-hand-side clauses of the operator. A set of predicates for use as test predicates in the experimental domain are found in Appendix J.

The conflict resolution function is called by the predicate select_ops/2. The arguments are the conflict set and a subset selected for execution. In the present implementation, the work of resolving conflicts among the matched instantiations is performed by making sure that only one act an any conversational level is chosen. This choice was made (1) in the interests of simplicity, and (2) as a way of expressing, as observed in the protocol discourse, that the conversants utterances generally focus on one theme. A problem with this function as implemented is that it simply uses rule order to determine which of the acts in any given level should be executed. This is an area where meta-rules for conflict resolution, representing processes for determining intentional choices, would be of value.

It would also be interesting to relax this constraint and see what happens if, for example, multiple domain-level acts were permitted simultaneously. This might reproduce the indirect speech-act behavior observed in the protocol at Ai2. In the present function, having reduced the conflict set in this manner, each act is assigned a priority corresponding to its level. Currently, the system chooses the lowest-level act as the most important. The rationale for this design decision was to preserve to coherence of the flow of the discourse, keeping up the feedback, even if it meant that domain-level actions were sometimes deferred. This reflects the behaviors observed in the protocol, where the confirmations of mutuality of knowledge continue even as a conversant is re-evaluating his or her domain goals. Finally, the conflict resolution function makes sure that the selected instantiations of operators have actions and effects which are consistent with each other. That is, one operator should not be adding a state which is being deleted by another. Only those operators whose effects are consistent with those of the priority operator are selected. The implementation of this function is reasonably straightforward; it is the design choice underlying it which still needs thought. It turns out that some operators will take an action—thus adding it to the conversant's own knowledge base—and then immediately delete it because a record of the action turns out not to be needed in active memory. The operator give_turn_1 (listed in Appendix H) is an example of this. In other cases, variations in belief states may cause apparent inconsistency to be found where the operators are really in substantial agreement. For example, an act added as a goal might also be added as a known act. Both of these problems would be minimized or eliminated by expansion of the system's capability to reason about oppportunistic action and the conversant's own intentionality.

The driver for the rule-based system runs the match-resolve-execute cycle for one conversant and then the other, using the same cycle number. Actions taken by the first agent are not used by the second agent in processing on the same cycle. This was implemented via the definition of active memory as those states which had been added *before* the current cycle. Drivers for single-cycle, multiple-cycle, and indefinite execution are available. These are run_single_cycle, run_to_cycle, and run, respectively. The indefinite execution halts when the system completes a cycle in which neither conversant has had any rule fire.

The I/O clauses create two output forms, a short trace and a full trace. The short trace is displayed on the screen as the system runs and is also incrementally appended into a file. The short trace shows which operators fired and what actions were taken. A short trace of a simulation of the protocol described in Chapter V is set out in Appendix E. The full trace is incrementally appended into a different file. This trace contains a cycle-by-cycle listing of active memory and full representations of all of the operators executed on each cycle. A full trace of the simulation protocol is set out in Appendix F.

The development tools for the system include clauses to check matching of a given operator for a conversant, to display active memory, to display other states in memory for other cycles, to display an operator, to run a cycle with full output to the terminal, to reinitialize memory and the operators, to add and delete states in the knowledge base, and to back the system up to a previous cycle state. These tools proved useful in both refining the operators and simulating miscomprehension by changing states in memory.

Implementation of the Model as Rules

In building a computational system which models the meta-locutionary acts, one might begin by recognizing that there is a correspondence between the action-producing rules and the acts themselves. That is, the firing of a rule which has as its consequence an act represents an actualization or embodiment of the situated act. There might be, though, situations which produce the same sort of act which differ by more than mere instantiation of their premises; wholely distinct sets of premises might cause the taking of the same act. As a result, acts should typically be represented by a corresponding *set* of rules (or operators, as they are called in the theory) rather than a single rule. In the particular system I have built, the rules create an instantiation of the act as a record of its being taken.

Through the process of developing the rules, intermixture of actions of different levels in the same rule was reduced and eventually eliminated. That is, rules represent operators at specific levels of the discourse hierarchy. The effect of one utterance embodying acts of different levels was produced through refinement of the left-hand-sides of the rules for the different-level operators. For example, an instantiation of the rule do_request, an information-level rule, is typically accompanied by an instantiation of one of the give_turn rules, which are turn-level rules.

Meta-Locutionary Rules

The simulation modeled meta-locutionay operators for turn-taking and utterance expression. The operators were written as rules in the turn-taking and information classes, respectively. For both levels, there were three basic types of rules: reasoning about intention, doing an act, and recognizing an act by the other agent. Figure 15 shows a rule for the act of assertion and Figure 16 shows a rule for recognizing an assertion by the other agent. The complete set of meta-locutionary rules used in the simulation is presented in Appendix H.

The rule shown in Figure 15 has one action, namely the asserting, and changes the belief state of the triggering clause from goal_true to true (via automatic posting of the action into the conversant's own active memory). The rule in Figure 16 has no action. It captures part of the process of reasoning about the other agent's knowledge. The effect of executing an instantiation of assertion_received_1 is to add to the conversant's active memory a state which represents the conversant's belief as to the belief of the other agent. This rule is naïve, because it is possible that the other agent is not telling the truth, but as no one in the protocols was found to have intentionally not spoken the truth, a more sophisticated rule was not needed for the simulation.

```
op(          % do_assert
 if([ conversants(Me,Other),
    act(Me,assert(Info,Other),goal_true),
    believe(Me,turn(Me),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(Me,assert(Info,Other),true) ]),
 effects([ del(act(Me,assert(Info,Other),goal_true)) ]),
 atts([ name(do_assert),
    level(information) ]) ).
```

FIGURE 15. Meta-locutionary rule do_assert. The rule is represented as a Prolog clause. Comments are marked by the '%' symbol. There are three 'if' clauses for the operator; these are are matched against a conversant's active memory. There is only one 'action' clause; it is added to the memory of both conversants.

```
op(          % assertion_received_1
 if([ conversants(Me,Other),
    act(Other,assert(believe(Other,Info,Belief_value),Me),true),
    believe(Me,turn(Other),mutually_known_true) ]),
 not([ believe(Other,Info,Belief_value) ]),
 test([]),
 action([]),
 effects([ add(believe(Other,Info,Belief_value)) ]),
 atts([ name(assertion_received_1),
    level(information) ]) ).
```

FIGURE 16. Meta-locutionary rule assertion_received_1. The rule is represented as a Prolog clause. Comments are marked by the '%' symbol.

The simulation showed that the number and complexity of rules needed to handle these basic functions is quite small. In all, 15 meta-locutionary operators were used in the simulation. Of course, these rules represented only two meta-locutionary conversational levels in addition to domain-knowledge levels. The other conversational levels implied in Figure 2 would have corresponding sets of additional operators in a more complete conversational simulation. Thus to the extent that additional skills for interaction are needed for coherence of more complex conversation, many more new operators might have to be added. The operators developed in this dissertation constitute perhaps 35 percent of the operators at their own conversational level. Accordingly, they certainly constitute no more than 10 percent of the total number of conversational operators for all meta-locutionary conversational levels from turn-taking up.

Domain Rules

Representing the illocutionary knowledge of the conversants turned out to be a more difficult task than representing the meta-locutionary knowledge, even in a such an apparently simple domain. Fifteen meta-locutionary rules were used in the simulation. Eighteen domain rules were written, of which 16 were used in the simulation whose trace is presented here. Again, there were rules for reasoning, taking, and recognizing acts. Figure 17 shows an example of a domain rule which reasons about acts to be taken. The rule, confirmed_next_letter, is instantiated if the agent (1) is trying to confirm mutual knowledge of a subsequence, (2) is trying to confirm mutual knowledge of a letter (presumably in that subsequence), and (3) now believes that the letter is, in fact, mutually known true. As a result, this operator will then remove the goals about the current letter and establish goals for the next letter to be confirmed. If there had been no next letter (i.e., if this had been the last letter in the subsequence), the test clause would have failed; I expect that the rule confirm_last_letter would then have been instantiated instead.

The biggest problem with the domain rules was prevention of repetitious and redundant instantiations. This was caused by the system's design using the situated action approach rather than a stack-based planner. That is, if a rule could be executed on one cycle, it would be likely to fire on the next cycle as well. This problem was tempered through the use of not clauses, which took account of the operator's own history in posting to the knowledge base. Such extensive use of nots, however, seems (1) unduly cumbersome and (2) unrealistic as a representation of human memory. Much of this process could be moved to the conflict resolution process if that process could use

```
op(          % confirmed_next_letter
  if([ conversants(Me,Other),
    act(Me,confirm_mutual(subsequence(S_index,List)),goal_true),
    act(Me,confirm_mutual(next_letter(letter(Index,Letter))),goal_true),
    believe(Me,next_letter(letter(Index,Letter)),mutually_known_true) ]),
  not([]),
  test([ get_next_letter(Index,List,New_letter) ]),
  actions([]),
  effects([
    del(act(Me,confirm_mutual(next_letter(letter(Index,Letter))),goal_true)),
    add(act(Me,confirm_mutual(next_letter(letter(Index,Letter))),true)),
    del(believe(Me,next_letter(letter(Index,Letter)),mutually_known_true)),
    add(believe(Me,next_letter(New_letter),true)) ]),
  atts([ name(confirmed_next_letter),
    level(domain) ]) ).
```

FIGURE 17. Domain rule confirmed_next_letter. The rule is represented as a Prolog clause. Comments are marked by the '%' symbol.

meta-logic to take account of the agent's intentional state. In other words, although an agent *could* repeat an action indefinitely, the fact that the agent has already performed an action usually means that the agent does not consider doing it again; changes in circumstances (which could include lack of response) might cause an action to be repeated. Thus as a strategy for conflict resolution, it might be possible choose actions on the basis of what the agent believes it is currently doing in the conversation. This implies that the conflict resolution strategy would have to be tied to the agent's history.

Another helpful approach to the control issue would be the addition of additional meta-rules which specifically had memory maintenance responsibilities. Thus, for example, a representation of focus structure could allow the operation of rules which removed facts from active memory when they pertained to a matter not in focus; other rules would restore facts to active memory when their subject again became the focus of the conversation.

Results of the Simulation

After development of the rule set, the system was used to simulate the conversation reported in the protocol analyzed in Chapter V. The initial state of the system is listed in Appendix K. In brief, for each agent the initial state identified the conversants, set up goals of confirming mutually their respective sequences, identified the first subsequence, and set an initial turn state. In addition, Adam's initial state included an act Barney's giving the turn to Adam; this brought the simulated conversation up to the point of A's act Ai1, and the simulation proceeded from there.

Simulation of Experimental Protocol

The first simulation looked at what the agents' conversation would be like in the absence of the misconstruals caused by homophony observed in the experimental protocol. Short and full traces of this simulation are presented in Appendices E and F. Figure 18 shows the first two full cycles of the short trace.

For cycle 1, the trace shows Adam reasoning that he should try to request the next subsequence. (Recall that A's first letter was a blank.) At the same time, Adam takes an act: acknowledgment of Barney's initial-state act of giving Adam the turn. Simultaneously, Barney is reasoning that he should try to confirm the next subsequence of his sequence. (B's sequence began with a letter.) For cycle 2, Adam, using his turn, takes an information-level act: requesting that Barney assert the first subsequence because he, Adam, has a blank. At the same time, an operator at the turn level fires, taking the act of giving Barney the turn. This is, clearly enough, a salutary effect, as Adam would want Barney to be able to respond to his request. Meanwhile, Barney is still reasoning about

```
adam's acts for cycle 1:
 goal_request_next_subsequence —
 acknowledge_my_turn —
    act(adam,acknowledge_turn(turn(adam),mutually_known_true),true)

barney's acts for cycle 1:
 goal_confirm_next_subsequence —

adam's acts for cycle 2:
 do_request —
    act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
         barney),true)
 give_turn_3 —
    act(adam,give_turn(barney),mutually_known_true)

barney's acts for cycle 2:
 goal_confirm_next_letter_1 —
 others_turn_acknowledged —
```

FIGURE 18. Trace of simulation of protocol. This figure shows the first two complete cycles of a multi-cycle interaction. The reasoning and acts of each conversant are considered to be simultaneous with those of the other conversant for a given cycle. The first level of indentation presents the names of the operators which were executed on the that cycle. If any acts were take, the acts follow the dash after the operator at the second level of indentation.

what to do next. He realizes that he will need to confirm the next (which in this case is the first) letter of the subsequence. At the same time, he recognizes that Adam has the turn on this cycle. At this point, the simulation has progressed up through act Ai2. In later cycles, Barney then asserts that the first letter is "I." Without the miscommunication engendered by homophony, he then requests that Adam tell him the second letter, giving Adam the turn. Adam does so. The agents then go on to begin to confirm the next subsequences. At this point, as can be seen in the traces in Appendices E and F, the conversation begins to break down. At cycle 19, the conversation stops. This failure is attributable to (1) the lack of domain rules for proper handling of the confirmation of individual letters, and (2) the lack of rules for agreement on which subsequence is being looked at. With the refinement and addition of these rules, it appears that the conversation would have continued until the sequences were mutually confirmed. As it was, the simulation was able to get through the first three or four exchanges in a meaningful (to us), plausible way.

Control Issues

The results of this simulation indicate that meta-locutionary operators can plausibly provide the control processes which lend coherence to mixed-initiative discourse. The traces of the simulation show the agents requesting, taking, giving, and holding turns as needed to accomplish their domain goals. This control was accomplished without the use of semaphores.[17] Rather, the agents had only their own understanding of what they believed the state of the conversation to be. Additionally, the simulation showed situated action being used successfully to drive the agents' responses. Until the exchanges began to break down due to domain-rule failures in the later cycles, a form of global coherency in the conversation was achieved entirely through the use of local, context-driven operators.

[17]A semaphore is a protected variable whose value can be accessed only by two indivisible operations. They are used in computing systems with multiple processes (1) to ensure mutual exclusion and (2) to implement process synchronization (Deitel, 1984).

It is possible that the turn-taking operators could create a state in which the agents repeatedly try to take or give turns, thus finding themselves in a loop. However, this sort of thrashing did not occur in any of the simulation runs in this experiment. To be certain of avoiding thrashing, more sophisticated use of the conversational history would be needed. If the agents had memorial operators which checked for fruitless repetition in turn-taking, they could cause the agents to, in effect, step back and try something else. Such control would, in effect, be meta-meta because it causes changes in the conversation's processes of control. The major point here is that to avoid ineffectual actions, the agent should have operators which reflect knowledge about the relationship between action patterns and intention.

The converse situation of thrashing is also a concern. In the simulation, each agent expects that if they take an action that the other agent will eventually react. For example, if an agent makes a request, they will wait for the other agent to respond. Thus if the other agent does not respond then the conversation will come to an unintended end. This situation is made possible mostly by lack of temporal awareness in the simulation. That is, none of the operators has an explicit representation of or responsibility for the passage of time. To overcome this problem of stalling, then, the system ought to have operators which note failure of timely response. Such effects were in fact observed in the protocols. For example, a reasonable interpretation of A's illocutionary act at Ai4 (in Figure 13) is that A has waited for a response to her request-act Ai2; when B fails to make a timely response, A then takes Ai4 to renew the request by trying to repair a problem which she infers B must have encountered in interpreting Ai2.

Simulation of Miscomprehension

The rule-based system was also used as an experimental environment for simulating the miscommunication observed in the protocol. The system was given the same initial state, and ran through the end of cycle 4. At this point, Adam's knowledge base was modified by changing the substance of B's act (as indicated in the discussion in Chapter 5 with respect to act Bi6). The simulation was then resumed. As it turned out, the simulation proceeded in a manner virtually identical to that reported above. The principal reason for this appears to be that Adam had no rule available analogous to the R1 operator of Appendix D, which would have permitted the agent to repeat its request as an explicit act. In the absence of operators which explicitly concern actions of correction and repair, it appears that the simulation will not produce remedial feedback. Accordingly, one might quite reasonably conclude that the repair phenomena reported in the socio-linguistic literature are the result of meta-acts specialized for this purpose. In other words, conversational repairs can be seen as constituting a domain in which persons often have tasks; as a consequence, conversants develop domain-specialized skills for particular kinds of repairs. The lack of operators representing these skills in the simulation left the agents unable to effect direct repairs.

Contrasts with Power's System

As Power (1979) observed, the real achievement of his conversational simulation between the robots John and Mary was that it was able to represent the point of an utterance. His system represented a major advance in this area, in that he was able to produce a form of coherent interaction based on recognition of illocutionary meaning. The current research also represents the illocutionary meaning of actions but does not follow Power in producing English text. However, the simulation reported here is a significant advance in three principal respects First, the simulation models control: the agents control and modify the flow of their own interaction. Power's robots relied on an executive to allocate control between them. This meant that their turns were explicit and did not have to be recognized. Additionally, they were unable to modify their interaction because the flow of the robots' conversation was basically pre-planned.

Second, the agents in the meta-locutionary simulation use locally situated acts to produce, without top-down planning, globally organized interaction. Power's robots had control stacks that recorded and executed conversational planning procedures. Indeed, the robots explicitly discussed their actions in terms of plans and intended plans. In the meta-locutionary simulation, the agents are guided by their intentions but do not have to be so explicit about developing plans; they are able to react more flexibly to context.

Third, the simulation models more realistic conversation instead of idealized speech. Power's work was directly in the tradition of speech-act theory. The conversations he modeled were fictional, idealized interactions between robots. The robots' acts were accordingly stylized. In contrast, the current work attempts to model the kinds of exchanges actually observed in human-human interaction. The acts used by the agents correspond to acts obtained from analysis of the experimental protocols.

Limitations and Extensions

The simulation as implemented had a number of limitations. Here I describe those limitations and suggest ways in which the system could be improved and extended. The limitations include over-specificity in matching operators against the agents' memory, separation of different cognitive processes into different cycles of the rules system, and a simplified representation for time. Going beyond the specifics of the implementation of the simulation, it would be possible to add other conversants and to push the notion of meta-locutionary action to include processes of lexical choice.

Implementation Issues

I begin this review of the simulation's limitations by examining the implementation of the of the simulation as a rule-based system. One problem, then, with the computational implementation so far is over-specificity in the operator-matching process. For example, operators may match and execute because a clause just happens to hanging around from some long-previous part of the conversation. The rules were written in such a way that they tried to prevent this problem by consistent deletion of states from active memory. This technique, though, suffers from drawbacks similar to those encountered with coding "not" clauses to prevent repetitious execution. This suggests that the conversant's models should be structured to provide appropriate contexts for matching. Such structure may arise from the multi-layered, hierarchical model of interaction presented in Figure 2. At the same time, there may be discourse-segment structure more closely allied with domain structure, along the lines of the focus spaces developed by Grosz (1981). This could be handled either by increasing the complexity of the key into the database or by adding cross-state references. It may be that explicit structures for focus are required. In the simulation, a lack of focus was most acutely noticed in the divergence of the agents' models of the concept of next subsequence: the agents had different patterns of blanks and letters, and so they divided their sequences into correspondingly different subsequences.

Another problem involved the design choice to spread the reasoning process over multiple cycles. This was intended to correspond to processing times for mental processes of conversants. In some cases this proved reasonable. In others, however, the multiple cycles needed for recognition, reasoning, and then action caused the agents to continue behaviors that in the protocols are immediately suspended. This suggests that the rules should be typed not only for level but for function as well. That is, rules would be of recognition, reasoning or action type. Then for each major cycle, the rule-based system would first fire each set of function-type rules in order, with later sets able to use the results of the earlier ones. This would necessitate reworking of the system for inter-agent simultaneity, but this should not be a difficult problem. In turn, if the temporal cost of the operators could be declared, then the system could keep track of inter-cycle dependencies in a more sophisticated manner than that currently used.

A third problem with simulation also has to do with time. In the system as implemented, all acts take unit time. For complex phenomena, this assumption is probably unwarranted. For example, uttering "O" may possibly be a unit act; reciting a poem surely is not. If the system is to represent processes like interruptions, then, there will have to be an explicit account of the temporal constraints on the physical processes of speech and understanding. Complex utterances will have to be produced over a sequence of cycles. At the limit, this could involve phoneme-by-phoneme (or at least word-by-word) production, and would constitute a logical extension of the meta-locutionary theory to speech production. This would also enable representation of self-correction. Absent an adequate representation for temporal extent, then, it will not be possible to build an adequate model of interruption or self-correction.

Possible Extensions

I turn now to issues beyond the those peculiar to the simulation's implementation. The system is not inherently limited to two conversants. The predicates representing the acts were developed so as to permit multiple conversants. For example, acts like give_turn, and do_request have slots for the "receiving" agent. Changing the rule-based system's driver clauses for multiple agents would be trivial. However, multi-agent protocols, even in the sequence-recollection domain, have proven harder to transcribe and map into acts.

The system could also be extended to encompass issues of lexical choice. In its current form, the model does not address the problem of transforming illocutionary acts into locutions; the acts are communicated directly, without being expressed as specific verbal and nonverbal language. If the use of the system is broadened to include human agents interacting with the simulation agents, such issues will have to be resolved. What sort of solutions would be reasonable? Straight act-to-utterance mappings would not be a good choice. While some physical actions have societally conventional "emblematic" meanings (Ekman, 1980), no action has meaning without context (Birdwhistell, 1970). That is, while under the right circumstances raising the shoulders together is understood as a physical lexeme generally called a shrug, there cannot be any universally appropriate action for conveying a particular meaning usually embodied by a shrug. Thus for meta-locutionary acts that are normally expressed through non-verbal channels, the system cannot simply look up *the* turn-holding behavior in some table; rather, an expression must be chosen which the context will then provide the needed meaning. Indeed, the problem of lexical choice can viewed as a conversational level in itself: the idea of meta-locutionary processes then might be extended to encompass operators which produced the expression of specific lexemes—verbal and nonverbal—as their acts.

CHAPTER VII

CONCLUSION

The results of this study, stated broadly, suggest that it is feasible to extend speech-act analysis to meta-locutions, that humans use feedback to maintain coherence, and that irregularities in speech are clues to understanding rather than noise to be cleaned away. A computational model of meta-locutionary acts was developed and validated through both a protocol study and a rule-based simulation.

Summary

This dissertation began by posing a question which I called the integration problem: how can the speech-act theory of conversational action be reconciled with the sociological view of the irregularities of actual interactive discourse? In response, I suggested that conversants maintain a shared model of their conversation; that is, they are jointly creating a single conversation and each conversant has a model of the conversation they believe themselves to be creating. In light, then, of the coöperative, feedback-infused, locally-controlled nature of conversation, I suggested that the conversational repairs which maintain the shared model can be seen as acts akin to speech acts, except that they specifically affect the process of the conversation itself rather than more general aspects of the world related, for example, to conversants' underlying needs.

The view of conversational coherence as being produced by meta-acts of the conversants establishes the goal of the research reported in this dissertation: trying to identify and to model computationally a set of acts relative to these sub-sentential levels of conversation. I then described a set of conversational effects, as reported in the sociological literature, which formed a basis for ascribing meta-acts to the conversants. I presented an experimental technique to observe these acts.

I applied the theory of meta-locution to the protocols which I had obtained. That is, given the theory, did it and account for the interaction observed in actual conversations? The theory suggested that meta-locutionary acts—acts by a conversant which control the process of the conversation—could be modeled at several different conversational levels, ranging from domain actions to turn-taking. In the case of turn-taking specifically, for example, the set of directive acts included hold-turn, give-turn, take-turn, and accede(turn). The interaction observed between the conversants could be explained computationally in terms of this set of meta-locutionary acts. Examples of operators were presented which, given the state of a conversant's model of the conversation, would account for the observed acts. In three specific cases, I used the meta-locutionary analysis to explain otherwise anomalous utterances. The protocol analysis also confirmed the observation that humans use feedback to maintain coherence. Indeed, most of the acts accounted for in the protocol are not domain-level acts but rather concern and affect the control and comprehension of the conversation itself. It is also true that the conversation obtained in the protocol contains all kinds of interruptions and sentence fragments. The meta-locutionary analysis of the protocol evidence is consistent with a view that each utterance, however fragmentary, has meaning in the conversational context. In understanding human communication, the individual fragments of the conversation are valuable as clues to coherent interpretation. I note that in the observed interaction, the production of utterances is inherently tied to understanding of the other conversant's utterances. Moreover, a number of the utterances are contemporaneous, suggesting that timing of utterance production is a mutually determined process, as might be expected in a coherent mixed-initiative dialogue. This suggests that in analyzing interactive discourse, language production and language understanding cannot be separated.

The computational model developed through the protocol analysis was tested through a rule-based simulation. Note that the simulation did not constitute the theory; rather, the theory was partially implemented as a rule-based system to evaluate its adequacy for computer-based interaction. The simulation presented in Chapter VI showed that meta-locutionary operators can control mixed-initiative discourse in a manner that resembles the kind of interaction observed in natural conversation. Meta-locutionary acts specifically modeled in the simulation included the requesting, taking, giving, and holding of conversational turns. The simulation also showed situated action producing a form of global coherency through the use of local, context-driven operators.

Open Issues

The research presented in this dissertation represents a set of first steps in understanding the control of mixed-initiative discourse. The central thesis of accounting for coherence through meta-locutionary acts appears well-founded. That the particular acts proposed here represent the ultimate expression of the theory seems less certain. The simulation study made clear the need for specialized operators (and presumably acts) dealing with correction and repair. Other classes of acts will probably emerge. Aside, though, from the obvious future work in this area involving the expansion and refinement of the acts, and aside from the design issues discussed with respect to the implementation of the simulation, there are other issues to address. Some of these issues are representational, and have to do with understanding what we are observing in conversations recorded in protocol studies. Other issues arise with respect to what other kinds of human-human interaction might this methodology be applied.

Representation Issues

With respect to the process of transcribing and encoding the protocols, there remain aspects of the conversations which are as yet unrepresented. First, pauses, while implicitly represented in the time-line transcript, are not specifically noted as either things or acts. It is possible that pauses may, in and of themselves, be acts. The interpretation of pauses as possible acts seems subjective in the extreme, however. Chafe (1986) suggested, in the context of generation, that pauses correspond to the process of memorial retrieval; that is, concepts have activation levels which determine the retrieval costs for related concepts. According to Chafe, patterns in discourse are shaped by the speaker's costs of retrieval corresponding to active, semi-active, and inactive concepts. In any event, the problem of accounting for pauses remains unsatisfactorily addressed by the meta-locutionary model.

Another problem arises out the subjective nature of the coding of acts from the utterances, which is necessarily a process of interpretation. In a qualitative study in which rules for coding acts from lexical representations have not been expressed, it falls upon the coder to make a subjective interpretation of the act in the context of the conversation. In the case of the protocol analyzed in this dissertation, this process has been an iterative one, with successive refinements and alterations of the encodings of the acts and the states in order to produce a rational account of the observed conversation. Does not this process result in an encoding for the conversation which *must* work, under the circumstances? The answer to this question is in three parts. First, although the acts are motivated by accounts in the socio-linguistic literature, the development of the specific acts and their computational representations is the express goal of this research. That is, this process is the one which Cohen (1984) proposes. Second, the set of acts, their representations, and the operators, are validated by application in the simulation. Third, there is a need for future work in applying the acts developed on the basis of the protocol reporter here to other protocols. If effectiveness outside of the context of their development can be observed, then the acts may reasonably considered valid.

Other Voices, Other Rooms

The theory may also have application for other sources of human interaction and other contexts. Clearly, the theory of meta-locutionary acts could be applied in other domains. If the theory is true, then meta-locutionary acts should be consistent among different speakers of a given language; otherwise we would find it hard to converse at all. Moreover, the acts should be fairly domain-independent. The dissertation research looked at two different pairs of speakers, both in the sequence-recollection domain. The main obstacle to application of the theory to other domains is difficulty of representing the domain knowledge with sufficient exactness to permit the simulation to run. The complexity of even the sequence-recollection domain, especially as compared to the meta-locutionary knowledge, turned out to have been daunting. The problem is not that it is hard to find *some* representation which would permit a computer program to perform the domain task in the abstract. Rather, the problem is in linking the representation of the domain to mental states which can form the basis for linguistic action (Stucky, 1988).

Speech act theory has also been applied to communication within organizations (Winograd & Flores, 1986). There may meta-locutionary analogs for communicative control and coherence in this interaction. This may be actually be a fruitful domain because of the relatively formalized and statically represented nature of intra-organizational communication.

Cognitive Constraints

The protocol analysis and simulation suggest two areas of inquiry as to the role of cognitive constraints in conversation. Understanding these constraints would aid explication of the observed discourse and would immediately serve to improve communication through computer-human interfaces.

Language Units

One of the issues to which the simulation forced attention was that of the size of the units of communication. How much can people understand at once? For computers this is generally not a problem; if the agents were simply Prolog programs they could have just transferred their sequences to each other and merged them. Obviously, most people cannot do this; the protocol subjects certainly didn't. The human cognitive constraints which determine how much we can absorb at a time thus have consequences for the design of mixed-initiative interfaces. The computer program must accommodate the human's limited capacity. Conversely, there may be constraints on the human capacity for linguistic production. Systems which are receiving information from humans may be better equipped to understand the interaction if they can relate the size of the units of language produced to factors associated with the production process.

The variation in human skills for interaction is large (see e.g., Argyle & Cook, 1976). Presumably conversants take account of and adapt for these differences, which inject a correspondingly large measure of uncertainty into the interaction process: Is the other conversant really attending? Was that a nod? Variation in the size of the units of language which conversants can produce or understand creates uncertainty as to (1) the degree of understanding and (2) the degree to which conversants believe that they have been understood. These factors, then, suggest reasons why feedback is such an important part of conversation. Conversants are continually faced with issues of understanding which are not ordinarily resolvable through one-sided inference. Thus greater understanding of the limits of human cognitive capacity would better allow modelers of interaction to base operators on the underlying reasons for the use of meta-locutionary acts.

Tolerance of Uncertainty

Having observed that conversation is replete with sources of uncertainty, I now turn to the some of the issues of how conversants manage the uncertainty they encounter. The central question I want to ask is: How tolerant of uncertainty are conversants? The answer to this question has important implications for problems such as deciding when to initiate conversational repairs. In the model and simulation, uncertainty was reduced to qualitative simplifications such as true and mutually_known_true. Yet conversants apparently proceed with imperfect knowledge of the state of the conversation or, when faced with obvious differences in their conversational models, will continue without repair if the differences are not too great. How much imperfection is too much? How great a difference is too great?

In addition to uncertainty with respect to mutuality of knowledge, there is uncertainty as to whether purely mental perlocutionary effects have been achieved. If a tutor asserts some fact to a student is the tutor justified in assuming that the students now knows the fact? How well is well enough? These are issues which might be addressed by psycho-linguistic experimentation. Conversations could be induced in which responses indicating various degrees of comprehension are returned to the subjects.

Finally, if conversants are not certain of the other's knowledge, they are also uncertain as to their own knowledge—or at least their understanding of their knowledge. As they listen, conversants seem to assume that if they are momentarily off track they will eventually recover using later information. Thus they may continue to give affirming signals to the speaker despite imperfect understanding of what is being said. Again, this phenomenon seems to be a matter of gradation. The level of tolerable uncertainty probably varies with context. To what extent, then, are conversants apt to defer repair? In interacting with computers, the perceived (or real) bother of repair may lead people to defer it to point of irreparability; the program's limited ability to track the conversant's knowledge may have been quickly exhausted.

Modality

The bandwidth limitations of human-computer interaction appear to limit the richness and ease of meta-locutionary action. Differences in modality cause changes in interaction patterns (see e.g., Argyle & Cook, 1976; Chapanis, et al., 1972, 1977; Grosz, 1982; Cohen, 1984). The changes are not always straightforward:

> Under telephone or no-vision conditions, utterances are often shorter, some studies find more pauses, and less mutual influence, but there are *fewer* interruptions. There is some evidence of poorer synchronizing (more pauses), and for transfer of function to other signals (more attention signals). There is a reversal of a clear-cut expectation about interruptions, suggesting either that people learn to use different cues over the telephone, or that the shift of gaze cue is not very helpful. Evidently people either do not or cannot interrupt each other if they cannot see each other. (Argyle & Cook, 1976, pp. 163-164)

This evidence suggests that with present technology the human-computer interface is at an inherent disadvantage for mixed-initiative interaction. There are a number of approaches for possible solutions to this problem. The first approach would involve alternate lexicalization of meta-locutionary acts which are expressed in the physical channel. The idea here is to provide, through design, a set of verbal lexemes that correspond to physical kinemes. Unfortunately, the history of artificial languages in human use is dim. The languages which thrive are those created through conventional acceptance of negotiated units and meanings. This suggests a second but more difficult approach: provide the computer program with (1) the skills to negotiate the use of meta-locutionary interaction and (2) connections to the community of programs in which the language is being negotiated. Human beings would have, I surmise, great difficulty in developing language skills (and the language itself) if isolated from the general community of language users. If anything, computer programs are more susceptible to failure on this account because their language-learning skills are at best rudimentary.

Beyond alternate lexicalization, it may be that different modalities lead to the use of different meta-locutionary acts altogether. That is, conversants do not simply express the same acts in different ways; rather, they actually interact differently. This is, I think, a fair inference from the results described by Argyle and Cook (1976) and Grosz (1982). Some of the observed differences may not arise directly out of the limitations on the kinds of communication permitted. Instead, there may be second-order effects from, for example, declines in the speed of feedback. That is, feedback would still be possible through the available channels, but the process of conversion would be too slow. Such effects certainly might lead to significant changes in the naturalistic feel of the interaction. Furthermore, it may be the case that the feedback-based production processes are, through nature or through entrenched habit, dependent on prompt interaction for their effectiveness. In this case, substitute acts for feedback would have to account for speed of interaction. In computer-human interaction through graphically and aurally based interfaces, the ease and speed of feedback is asymmetric. The computer program can provide feedback much more rapidly than it can interpret it. This suggests that if meta-locutionary acts are to be useful for computer interfaces greater efforts should be made in the technology of physical input devices.

Humans and Computers

Over the course of the preceding chapters, I have tried to account for the differences between human-human and human-computer interaction. Many of these differences, I argued, can be explained by understanding mixed-initiative discourse as a multi-level process which uses meta-acts to produce coherence. In reaching these conclusions (and in applying them), though, we again face the limits of our knowledge of fundamental aspects of mind. In the case of intention, for example, the model presented in this dissertation achieves a reasonable level of success by ascribing intentionality to meta-behaviors. The upper levels of the conversational process rely on fairly diffuse intentions which are then translated into smaller and more more specific intentions. In this process, planning is not used (although it might be useful in generating the high-level intentions); rather, the intentional acts set up expectations which are taken into account as part of the conversational context for later action. But as I discussed in Chapter III, the limits of this approach are near. It may be that intention itself is not a useful concept for inducing action. Suchman (1987) suggested that plans are a post-hoc rationalization of the organization of behavior rather than the mechanism which produces it. How can we exclude, then, the possibility that intention itself is simply a post-hoc rationalization of the causes of behavior rather than the mechanism which leads to it? The problem, though, is that at this point we simply do not have an alternative theory of action. If not for intention, how else—or why else—would people do things?

Another fundamental limit arises out of what we perceive to be mixed-initiative interaction. In this dissertation, I have attempted to explicate mixed-initiative discourse through meta-acts. But if these acts are all co-temporally related behaviors at different conversational levels, the problem of mixing initiative also may arise at the meta-locutionary levels. I have suggested that the "lower" levels of conversational interaction represent a set of base cases for this recursion, but this cannot be concluded with confidence absent further research.

Yet even if we are reaching limits of knowledge, I hope that these limits are better defined and better illuminated as the result of this work. In the issues which I discussed, the fundamentals come down to the fact that research into human-computer interaction must tell us more about what it means to be a human being, what it means to be a computer, and what it means to interact.

APPENDIX A

PROTOCOL TRANSCRIPT

INDEX:	
T	time
Av	A's verbal
Bv	B's verbal
Ap	A's physical
Bp	B's physical
Ai	A's illocution
Bi	B's illocution

KEY:

thing	**direction**	**object**
h head	U up	A conversant
e eyes	D down	B conversant
a arm	L left	O object
ah hand	R right	E experimenter
r right		
l left		
f finger		

action	**notation**
W forehead-wrinkles	() previous state
C forehead-clear	... duration
O mouth-open	
S mouth-shut	

```
T   00.0      00.2      00.4      00.6      00.8      01.0      01.2      01.4      01.6      01.8...
Av                            blank................................
Ap (turning-hUBeB) hDB...
Ai                           Ai1Ai2

Bv 'kay....................
Bp (hAeA,rlhU)    rlhD,together,leaning-back     mO.......
Bi Bi1                 Bi2                                         Bi4
```

Bi1:{indicating that conversation can start}

Bi2: give-turn(B,A)

Ai1: take-turn(A)

Ai2: inform(A,B,{?X = doesn't have first letter}

Bi4: take-turn(B)

```
T   02.0      02.2      02.4      02.6      02.8      03.0      03.2      03.4      03.6      03.8...
Av                         is...the.....first...one....so....what...was.the.first
Ap            eBlink
Ai            Ai3             Ai4                                              ....Ai5

Bv                                                                     "O"...........
Bp                                                  mO
Bi                                                  Bi5                     Bi6
```

Ai3: inform(A,B,comprehend(A,take-turn(B)))

Ai4: clarify(A,?X); request(A,B,inform(B,A,{?Y = first letter}))

Bi5: take-turn(B)

Bi6: inform(B,A,?Y = first letter is "O")

Ai5: repeat(A,request(A,B,inform(B,A,{?Y = first letter}))), give-turn(A,B)

```
T  04.0   04.2   04.4   04.6   04.8   05.0   05.2   05.4   05.6   05.8...
Av ....one.......                                   OK..........
Ap                                                       eBlink
Ai                                                  Ai6  Ai6a

Bv                             "O"......                       an'.I.....
Bp                  eBlink
Bi                  Bi7        Bi8                             Bi10
```

Bi7: confirm-mutual(request(A,B,inform(B,A,{?Y = first letter}))))

Bi8: repeat(B,inform(B,A,?Y = first letter is "O"))

Ai6: confirm-mutual(A,B,?Y)

Ai6a: give-turn(A,B)

Bi10: inform(B,A,{?Z = second letter is "I"})

```
T  06.0   06.2   06.4   06.6   06.8   07.0   07.2   07.4   07.6   07.8...
Av            "I"..................          "S"...............   "U"....
Ap         hNod                        hNod/eBlink            hNod/eBlink
Ai         Ai7Ai8                      Ai9   Ai10                  Ai11

Bv ...                                                            "U"..
Bp                                hNod,hNod/eBlink.                hNod
Bi                                Bi10 Bi11                        Bi12
```

Ai7: acknowledge(A,B10)

Ai8: confirm-mutual(A,B,?Z)

Bi10: confirm-mutual(B,A,?Z)

Ai9: take-turn(A)

Bi11: accede(B,take-turn(A))

Ai10: inform(A,B,{?M = next letter is "S")

Ai11: take-turn(A), inform(A,B,{?N = next letter is "U"}

Bi12: accede(B,turnA) [implicitly accepting ?M as mutual]

```
T  08.0   08.2   08.4   08.6   08.8   09.0   09.2   09.4   09.6   09.8...
Av .........                          "T"..............
Ap                                         hR,hW........................
Ai            Ai12                    Ai13 Ai14

Bv ..........."R".................                     "T"..............
Bp            hNod                    hNod           hnod/eBlink.........
Bi            Bi13                    Bi14           Bi15Bi 16
```

Ai12: take-turn(A), inform(A,B,{?O = next letter is "R"})

Bi13: accede(B,take-turn(A)) [implicitly accepting ?N as mutual]

Ai13: take-turn(A), inform(A,B,{?P = next letter is "T"})

Bi14: accede(B,take-turn(A)) [implicitly accepting ?O as mutual]

Ai14: request(A,B,confirm-mutual(B,?P))

Bi15: take-turn(B)

Bi16: confirm-mutual(B,?P)

```
T  10.0   10.2   10.4   10.6   10.8   11.0   11.2   11.4   11.6   11.8...
Av     no...............                                   mine...went...
Ap     eBlink/hShake                                     eBlink
eD@lfs,
Ai     Ai15                                              Ai16Ai17

Bv                         I.had.a.."T".......
Bp .......                 hNod,hµNods.......            eBlink
Bi                         Bi17                          Bi18      Bi18
```

Ai15: deny(B,?P)

Bi17: repeat(B,Bi16)

Ai16: take-turn(A)

Bi18: acceed(take-turn(A))

Ai17: inform(A,B,{?Q = initial sequence})

Bi18: hold-turn(A)

```
T  12.0   12.2   12.4   12.6   12.8   13.0   13.2   13.4   13.6   13.8...
Av ....                          blank................. so..its......."I".
Ap lfs-riffle......    eUB                                     eD@fs,hD...
Ai                     Ai19    Ai20                     Ai21Ai22
Ai23

Bv                       "S".....“U"...."R"...."T".......
Bp                                                      eBlink........
Bi                       Bi19                           Bi20
```

Ai19: give-turn(A,B)

Bi19: inform(B,A,{?Q = intitial sequence})

Ai20: inform(A,B,?X)

Bi20: give-turn(B,A)

Ai21: inform(A,B,{A will discuss ?Q})

Ai22: hold-turn(A)

Ai23: confirm-mutual(A,B,{first letter is "I"} [wrong]

```
T  14.0   14.2   14.4   14.6   14.8   15.0   15.2   15.4   15.6   15.8...
Av ................."S"....................    "U"..................
Ap    eUBhU.                                              eBlink    eBlink
Ai    Ai24          Ai25                           Ai26   Ai27        Ai28

Bv
Bp
Bi
```

Ai24: check(A,B)

Ai25: confirm-mutual(A,B,{second letter is "S"})

Ai26: confirm-mutual(A,B,{third letter is "U"})

Ai27: check(A,B)

Ai28: hold-turn(A) [I'm thinking]

```
T  16.0   16.2   16.4   16.6   16.8   17.0   17.2   17.4   17.6   17.8...
Av                                    "T"....................."R"......
Ap ..........eDist                               eBlink,hLB,eB..
Ai           Ai29                     Ai30       Ai31           Ai32

Bv                                     puhbuh..
Bp                                                              hU.......
Bi                                    Bi21                      Bi22
```

Ai29: hold-turn(A) [I'm still thinking]

Ai30: confirm-mutual(A,B,{fourth letter is "T"})

Bi21: take-turn(B),request(B,A,attend(A,B))

Ai31: attend(A,B)

Ai32: confirm-mutual(A,B,{next letter is "R"})

Bi22: attend(B,A)

```
T  18.0   18.2   18.4   18.6   18.8   19.0   19.2   19.4   19.6   19.8...
Av ...........
Ap hD,hW..............................hU,hC               eBlink,eDist
Ai Ai33                                Ai34                Aii35

Bv
Bp hD.........    hNod/eBlink............
Bi Bi23           Bi2
```

Ai33: request(A,B,confirm-mutual(B,A,{next letter is "R"}))

Bi23: confirm-mutual(A,B,{next letter is "R"})

Bi24: repeat(Bi23)

Ai34: confirm-mutual(A,B,{next letter is "R"})

Ai35: hold-turn(A)

APPENDIX B

PREDICATE REPRESENTATIONS

Meta-locutionary illocutionary acts:

accede(<person1>,turn(<person2>))	<person1> lets <person2> take turn
acknowledge(<person1>,Act)	<person1> recognizes act
attend(<person1>,<person2>)	<person1> pays attention to <person2>
clarify(<person1>,State)	<person1> elaborates knowledge of state
comprehend(<person1>,State)	<person1> understands State is true
confirm-mutual(<person1>,<person2>,State)	<person1> says both conversants know that state is true
deny(<person1>,State)	<person1> denies the truth of state
give-turn(<person1>,<person2>)	<person1> gives turn to <person2>
hold-turn(<person1>)	<person1> holds on to turn
inform(<person1>,<person2>,State)	<person1> asserts to <person2> that state is true
repeat(<person1>,Act>)	<person1> repeats act
request(<person1>,<prson2>,Act)	<person1> requests <person2> to perform act
take-turn(<person1>)	<person1> takes turn

APPENDIX C

CONVERSATIONAL MODELS

Bi1:{indicating that conversation can start}
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))

Bi2: give-turn(B,A)
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
accedes(B,(turn(A)).

Ai1: take-turn(A)
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).

Ai2: inform(A,B,{?X = doesn't have first letter}
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).

Bi3: inform(B,A,comprehend(B,?X))
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).

Bi4: take-turn(B)
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).
wants(B,turn(B)).

Ai3: inform(A,B,comprehend(A,take-turn(B)))
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).
mutually-known(wants(B,turn(B))).

Ai4: clarify(A,?X); request(A,B,inform(B,A,{?Y = first letter}))
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).
mutually-known(wants(B,turn(B))).
wants(B,clarify(A,?X).

```
Bi5: take-turn(B)
not(know(A,<first letter>)).
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).
mutually-known(wants(B,turn(B))).
wants(B,clarify(A,?X).

Bi6: inform(B,A,?Y = first letter is "O")
wants(A,inform(B,A,<first letter>))
turn(A).
informed(A,B,not(know(A,<first letter>))).
mutually-known(wants(B,turn(B))).
wants(B,clarify(A,?X).
mutually-known(not(know(A,<first letter>))).
not(mutually-known(wants(A,inform(B,A,<first letter>)))).

Ai5: repeat(A,request(A,B,inform(B,A,{?Y = first letter}))),
give-turn(A,B)
wants(A,inform(B,A,<first letter>))
informed(A,B,not(know(A,<first letter>))).
wants(B,clarify(A,?X).
mutually-known(not(know(A,<first letter>))).
mutually-known(wants(A,inform(B,A,<first letter>))).
turn(B).

Bi7: confirm-mutual(request(A,B,inform(B,A,{?Y = first
letter}))))
wants(A,inform(B,A,<first letter>))
informed(A,B,not(know(A,<first letter>))).
mutually-known(not(know(A,<first letter>))).
mutually-known(wants(A,inform(B,A,<first letter>))).
turn(B).

Bi8: repeat(B,inform(B,A,?Y = first letter is "O"))
informed(A,B,not(know(A,<first letter>))).
mutually-known(not(know(A,<first letter>))).
mutually-known(wants(A,inform(B,A,<first letter>))).
knows(A,B,?Y = first letter is "O").
turn(A)

Ai6: acknowledge(A,?Y), confirm-mutual(A,B,?Y)
mutually-known(A,B,<first letter is "O">)
turn(A).

Ai6a: give-turn(A,B)
mutually-known(A,B,<first letter is "O">)
turn(B).
```

```
Bi10: inform(B,A,{?Z = second letter is "I"})
mutually-known(A,B,<first letter is "O">)
turn(B).
mutually-known(A,B,<second letter is "I">)
```

APPENDIX D

CONVERSATIONAL OPERATORS

Information/Reference-Level Operators

```
OPERATOR assert:
level:  information/reference

IF        wants(Me,assert(Me,Info,Other))
          turn(Me)
THEN      assert(Me,Info,Other)
EFFECTS   - wants(Me,act(Me,assert(Info,Other))


OPERATOR assertion-received:
level:  information

IF        act(Other,assert(believe(Other,Info),Me))
          turn(Other)
          not(believe(Other,Info))
EFFECTS   + believe(Other,Info)


OPERATOR do-request:
level:  information

IF        wants(Me,request(Me,Info,Other))
          turn(Me)
THEN      request(Me,Info,Other)
EFFECTS   - wants(Me,request(Me,Info,Other))


OPERATOR do-request-received:
level:  information

IF        request(Other,act(Me,Act),Me))
          wants(Me,act(Me,Act))
EFFECTS   - request(Other,act(Me,Act),Me))
```

Turn-Level Operators

```
OPERATOR acknowledge-my-turn:
level:   turn

IF        turn(Other)
          give_turn(Other,Me)
THEN      acknowledge_turn(Me,turn(Me))
EFFECTS   - give_turn(Other,Me)
          - turn(Other)
          + turn(Me)


OPERATOR acknowledge-others-turn:
level:   turn

IF        turn(Me)
          take_turn(Other)
          not(wants(Me,act(Me,Act)))
THEN      acknowledge_turn(turn(Other))
EFFECTS   - take_turn(Other)
          - turn(Me)
          + turn(Other)


OPERATOR recognize-my-turn-1:
level:   turn

IF        give_turn(Other,Me)
          turn(Other)
          not(request_turn(Me))
EFFECTS   - turn(Other)
          - give_turn(Other,Me)
          + turn(Me)


OPERATOR recognize-my-turn-2:
level:   turn

IF        give_turn(Other,Me)
          request-turn(Me)
          turn(Other)
EFFECTS   - turn(Other)
          - request-turn(Me)
          - give_turn(Other,Me)
          + turn(Me)
```

OPERATOR give-turn-1:
```
level:   turn

IF        wants(Me,act(Other,Act))
          turn(Me)
THEN      give_turn(Me,Other)
EFFECTS   - turn(Me)
          + turn(Other)
```

OPERATOR give-turn-2:
```
level:   turn

IF        request_turn(Other)
          turn(Me)
          not(wants(Me,act(Me,Act)
THEN      give_turn(Me,Other)
EFFECTS   - turn(Me)
          - request_turn(Other)
          + turn(Other)
```

OPERATOR give-turn-3:
```
level:   turn

IF        request(Me,Info,Other)
          turn(Me)
THEN      give_turn(Me,Other)
EFFECTS   - turn(Me)
          + turn(Other)
```

OPERATOR request-turn:
```
level:   turn

IF        wants(Me,act(Me,Act))
          turn(Other)
          not(request_turn(Me))
          (Act = assert(Me,Info,Other)) or (Act = request(Me,Req,Other))
THEN      request_turn(Me)
```

OPERATOR hold-turn:
```
level:   turn

IF        wants(Me,act(Me,Act))
          turn(Me)
          not(hold_turn(Me))
          (Act = assert(Me,Info,Other)) or (Act = request(Me,Req,Other))
THEN      hold_turn(Me)
```

APPENDIX E

SHORT TRACE OF SIMULATED CONVERSATION

```
adam's acts for cycle 1:
 goal_request_next_subsequence --
 acknowledge_my_turn --
    act(adam,acknowledge_turn(turn(adam),mutually_known_true),true)

barney's acts for cycle 1:
 goal_confirm_next_subsequence --

adam's acts for cycle 2:
 do_request --

act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),barney),true)
 give_turn_3 --
    act(adam,give_turn(barney),mutually_known_true)

barney's acts for cycle 2:
 goal_confirm_next_letter_1 --
 others_turn_acknowledged --

adam's acts for cycle 3:

barney's acts for cycle 3:
 goal_assert_next_subsequence_2 --
 recognize_my_turn_1 --

adam's acts for cycle 4:

barney's acts for cycle 4:
 do_assert --
    act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),true)

adam's acts for cycle 5:

barney's acts for cycle 5:
 asserted_next_subsequence --

adam's acts for cycle 6:

barney's acts for cycle 6:
 goal_request_next_subsequence --

adam's acts for cycle 7:

barney's acts for cycle 7:
 do_request --

act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),goal_true),adam),true)
 give_turn_3 --
    act(barney,give_turn(adam),mutually_known_true)
```

```
adam's acts for cycle 8:
  goal_assert_next_subsequence_1 --
  recognize_my_turn_1 --

barney's acts for cycle 8:

adam's acts for cycle 9:
  do_assert --
      act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true)
  request_turn --
      act(adam,request_turn,true)

barney's acts for cycle 9:

adam's acts for cycle 10:
  asserted_next_subsequence --

barney's acts for cycle 10:
  informed_of_next_subsequence_by_other --
  assertion_received_1 --

adam's acts for cycle 11:
  goal_confirm_next_subsequence --

barney's acts for cycle 11:
  goal_confirm_next_subsequence --

adam's acts for cycle 12:
  goal_confirm_next_letter_1 --

barney's acts for cycle 12:
  goal_confirm_next_letter_1 --

adam's acts for cycle 13:
  do_assert --
      act(adam,assert(next_letter(letter(1,r)),barney),true)

barney's acts for cycle 13:
  goal_confirm_next_letter_1 --
  request_turn --
      act(barney,request_turn,true)

adam's acts for cycle 14:
  give_turn_2 --
      act(adam,give_turn(barney),true)

barney's acts for cycle 14:
  regain_turn --
      act(barney,hold_turn,mutually_known_true)
```

```
adam's acts for cycle 15:

barney's acts for cycle 15:
 acknowledge_my_turn --
    act(barney,acknowledge_turn(turn(barney),mutually_known_true),true)

adam's acts for cycle 16:

barney's acts for cycle 16:
 do_assert --
    act(barney,assert(next_letter(letter(1,o)),adam),true)

adam's acts for cycle 17:

barney's acts for cycle 17:
 do_assert --
    act(barney,assert(next_letter(letter(1,s)),adam),true)
 give_turn_2 --
    act(barney,give_turn(adam),true)

adam's acts for cycle 18:
 acknowledge_my_turn --
    act(adam,acknowledge_turn(turn(adam),mutually_known_true),true)

barney's acts for cycle 18:

adam's acts for cycle 19:

barney's acts for cycle 19:
```

APPENDIX F

FULL TRACE OF SIMULATED CONVERSATION

Initial State:

```
barney: state(believe(barney,next_subsequence(subsequence(1,[o])),true),0,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(believe(barney,turn(adam),true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,give_turn(adam),true),0,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,
          blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,
          blank,blank,blank,g,o])),goal_true),0,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 1:

```
op(     % goal_request_next_subsequence
  if([ conversants(adam,barney),
      act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,
            blank,g,o])),goal_true),
    believe(adam,next_subsequence(subsequence(1,[blank])),true) ]),
  not([ act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
            barney),goal_true),
    act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
            barney),true) ]),
  test([]),
  action([]),
  effects([ add(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),
            goal_true),barney),goal_true)) ]),
  atts([ level(domain),
    name(goal_request_next_subsequence) ]) ).

op(     % acknowledge_my_turn
  if([ conversants(adam,barney),
    believe(adam,turn(barney),mutually_known_true),
    act(barney,give_turn(adam),true) ]),
  not([]),
  test([]),
  action([ act(adam,acknowledge_turn(turn(adam),mutually_known_true),true) ]),
  effects([ del(act(barney,give_turn(adam),true)),
    del(believe(adam,turn(barney),mutually_known_true)),
    add(believe(adam,turn(adam),mutually_known_true)) ]),
  atts([ level(turn),
    name(acknowledge_my_turn) ]) ).
```

barney's acts for cycle 1:

```
op(     % goal_confirm_next_subsequence
  if([ conversants(barney,adam),
      act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,
            g,o])),goal_true),
    believe(barney,next_subsequence(subsequence(1,[o])),true) ]),
  not([ act(barney,confirm_mutual(subsequence(1,[o])),goal_true),
    act(barney,assert(believe(barney,subsequence(1,_4140),true),adam),goal_true),
    act(barney,assert(believe(barney,subsequence(1,_4156),true),adam),true) ]),
  test([ not(o = blank) ]),
  action([]),
  effects([ add(act(barney,confirm_mutual(subsequence(1,[o])),goal_true)),
    add(believe(barney,next_letter(letter(1,o)),true)) ]),
  atts([ level(domain),
    name(goal_confirm_next_subsequence) ]) ).
```

```
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(act(barney,confirm_mutual(subsequence(1,[o])),goal_true),1,-1)
barney: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
barney: state(believe(barney,next_subsequence(subsequence(1,[o])),true),0,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(believe(barney,turn(adam),true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(adam),mutually_known_true),1,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),goal_true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 2:

```
op(    % do_request
 if([ conversants(adam,barney),
   act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
         barney),goal_true),
   believe(adam,turn(adam),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
         barney),true) ]),
 effects([ del(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),
         goal_true),barney),goal_true)),
   del(believe(adam,turn(adam),mutually_known_true)),
   add(believe(adam,turn(barney),mutually_known_true)) ]),
 atts([ name(do_request),
   level(information) ]) ).

op(    % give_turn_3
 if([ conversants(adam,barney),
   act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
         barney),goal_true),
   believe(adam,turn(adam),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(adam,give_turn(barney),mutually_known_true) ]),
 effects([ del(believe(adam,turn(adam),mutually_known_true)),
   del(act(adam,give_turn(barney),mutually_known_true)),
   add(believe(adam,turn(barney),true)) ]),
 atts([ name(give_turn_3),
   level(turn) ]) ).
```

barney's acts for cycle 2:

```
op(    % goal_confirm_next_letter_1
 if([ conversants(barney,adam),
   act(barney,confirm_mutual(subsequence(1,[o])),goal_true),
   believe(barney,next_letter(letter(1,o)),true) ]),
 not([ act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true) ]),
 test([]),
 action([]),
 effects([ add(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true)),
   add(act(barney,assert(next_letter(letter(1,o)),adam),goal_true)) ]),
 atts([ level(domain),
   name(goal_confirm_next_letter_1) ]) ).
```

```
op(     % others_turn_acknowledged
  if([ conversants(barney,adam),
     believe(barney,turn(adam),true),
     act(adam,acknowledge_turn(turn(adam),mutually_known_true),true) ]),
  not([]),
  test([]),
  action([]),
  effects([ del(believe(barney,turn(adam),true)),
     add(believe(barney,turn(adam),mutually_known_true)),
     del(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true)) ]),
  atts([ level(turn),
     name(others_turn_acknowledged) ]) ).
```

```
barney: state(believe(barney,turn(adam),mutually_known_true),2,-1)
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),goal_true),2,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),2,-1)
barney: state(act(adam,give_turn(barney),mutually_known_true),2,-1)
barney: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
            barney),true),2,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(act(barney,confirm_mutual(subsequence(1,[o])),goal_true),1,-1)
barney: state(believe(barney,next_subsequence(subsequence(1,[o])),true),0,-1)
barney:  state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
            goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
            blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(barney),true),2,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
            barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam:  state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
            blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam:  state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
            blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 3:

barney's acts for cycle 3:

```
op(     % goal_assert_next_subsequence_2
 if([ conversants(barney,adam),
      believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
           goal_mutually_known_true),
   act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
           barney),true),
   act(barney,confirm_mutual(subsequence(1,[o])),goal_true),
   act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),
   act(barney,assert(next_letter(letter(1,o)),adam),goal_true),
   believe(barney,next_subsequence(subsequence(1,[o])),true) ]),
 not([ act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),goal_true) ]),
 test([  get_subsequence(1,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
           subsequence(1,[o])) ]),
 action([]),
 effects([ del(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),
           goal_true),barney),true)),
   del(act(barney,confirm_mutual(subsequence(1,[o])),goal_true)),
   del(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true)),
   del(act(barney,assert(next_letter(letter(1,o)),adam),goal_true)),
   add(act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),goal_true)),
   del(believe(barney,next_subsequence(subsequence(1,[o])),true)),
   add(believe(barney,next_subsequence(subsequence(1,[o])),true)) ]),
 atts([ level(domain),
   name(goal_assert_next_subsequence_2) ]) ).

op(     % recognize_my_turn_1
 if([ conversants(barney,adam),
   act(adam,give_turn(barney),mutually_known_true),
   believe(barney,turn(adam),mutually_known_true) ]),
 not([ act(barney,request - turn,true) ]),
 test([]),
 action([]),
 effects([ del(believe(barney,turn(adam),mutually_known_true)),
   del(act(adam,give_turn(barney),mutually_known_true)),
   add(believe(barney,turn(barney),mutually_known_true)) ]),
 atts([ name(recognize_my_turn_1),
   level(turn) ]) ).
```

```
barney: state(believe(barney,turn(barney),mutually_known_true),3,-1)
barney: state(believe(barney,next_subsequence(subsequence(1,[o])),true),3,-1)
barney: state(act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),
          goal_true),3,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(barney),true),2,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 4:

barney's acts for cycle 4:

```
op(     % do_assert
 if([ conversants(barney,adam),
    act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),goal_true),
    believe(barney,turn(barney),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),true) ]),
 effects([ del(act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),
           goal_true)) ]),
 atts([ name(do_assert),
    level(information) ]) ).
```

```
barney: state(act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),true),4,-1)
barney: state(believe(barney,turn(barney),mutually_known_true),3,-1)
barney: state(believe(barney,next_subsequence(subsequence(1,[o])),true),3,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(barney),true),2,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 5:

barney's acts for cycle 5:

```
op(    % asserted_next_subsequence
 if([ conversants(barney,adam),
     believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
           goal_mutually_known_true),
   believe(barney,next_subsequence(subsequence(1,[o])),true),
   act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),true) ]),
 not([]),
 test([ length([o],1),
     length([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o],17),
   1 + 1 =< 17,
    get_next_subsequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o],
           subsequence(1,[o]),subsequence(2,[blank])) ]),
 action([]),
 effects([ del(believe(barney,next_subsequence(subsequence(1,[o])),true)),
   del(act(barney,assert(believe(barney,subsequence(1,[o]),true),adam),true)),
   add(believe(barney,next_subsequence(subsequence(2,[blank])),true)) ]),
 atts([ level(domain),
   name(asserted_next_subsequence) ]) ).


barney: state(believe(barney,next_subsequence(subsequence(2,[blank])),true),5,-1)
barney: state(believe(barney,turn(barney),mutually_known_true),3,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney:  state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
           goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
           blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(barney),true),2,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
           barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam:  state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
           blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
           blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 6:

barney's acts for cycle 6:

```
op(    % goal_request_next_subsequence
 if([ conversants(barney,adam),
     act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,
            g,o])),goal_true),
    believe(barney,next_subsequence(subsequence(2,[blank])),true) ]),
 not([ act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
            goal_true),adam),goal_true),
    act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
            goal_true),adam),true) ]),
 test([]),
 action([]),
 effects([ add(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
            goal_true),adam),goal_true)) ]),
 atts([ level(domain),
    name(goal_request_next_subsequence) ]) ).
```

```
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true), adam),goal_true),6,-1)
barney: state(believe(barney,next_subsequence(subsequence(2,[blank])),true),5,-1)
barney: state(believe(barney,turn(barney),mutually_known_true),3,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(barney),true),2,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 7:

barney's acts for cycle 7:

```
op(     % do_request
 if([ conversants(barney,adam),
   act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
           goal_true),adam),goal_true),
   believe(barney,turn(barney),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
           goal_true),adam),true) ]),
 effects([ del(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
           goal_true),adam),goal_true)),
   del(believe(barney,turn(barney),mutually_known_true)),
   add(believe(barney,turn(adam),mutually_known_true)) ]),
 atts([ name(do_request),
   level(information) ]) ).

op(     % give_turn_3
 if([ conversants(barney,adam),
   act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
           goal_true),adam),goal_true),
   believe(barney,turn(barney),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(barney,give_turn(adam),mutually_known_true) ]),
 effects([ del(believe(barney,turn(barney),mutually_known_true)),
   del(act(barney,give_turn(adam),mutually_known_true)),
   add(believe(barney,turn(adam),true)) ]),
 atts([ name(give_turn_3),
   level(turn) ]) ).
```

```
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_subsequence(subsequence(2,[blank])),true),5,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,give_turn(adam),mutually_known_true),7,-1)
adam: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),goal_true),
          adam),true),7,-1)
adam: state(believe(adam,turn(barney),true),2,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,next_subsequence(subsequence(1,[blank])),true),0,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 8:

```
op(    % goal_assert_next_subsequence_1
 if([ conversants(adam,barney),
     believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,blank,g,o]),
          goal_mutually_known_true),
   act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),
   believe(adam,next_subsequence(subsequence(1,[blank])),true) ]),
 not([ act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),goal_true),
   act(adam,confirm_mutual(subsequence(2,_506)),goal_true) ]),
 test([ get_subsequence(2,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,
          blank,g,o]),subsequence(2,[i,s,u,t,w])) ]),
 action([]),
 effects([ del(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true)),
   add(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),
          barney),goal_true)),
   del(believe(adam,next_subsequence(subsequence(1,[blank])),true)),
   add(believe(adam,next_subsequence(subsequence(2,[i,s,u,t,w])),true)) ]),
 atts([ level(domain),
   name(goal_assert_next_subsequence_1) ]) ).

op(    % recognize_my_turn_1
 if([ conversants(adam,barney),
   act(barney,give_turn(adam),mutually_known_true),
   believe(adam,turn(barney),true) ]),
 not([ act(adam,request - turn,true) ]),
 test([]),
 action([]),
 effects([ del(believe(adam,turn(barney),true)),
   del(act(barney,give_turn(adam),mutually_known_true)),
   add(believe(adam,turn(adam),mutually_known_true)) ]),
 atts([ name(recognize_my_turn_1),
   level(turn) ]) ).
```

barney's acts for cycle 8:

```
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
            goal_true),adam),true),7,-1)
barney: state(believe(barney,next_subsequence(subsequence(2,[blank])),true),5,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney:  state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
            goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
            blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(adam),mutually_known_true),8,-1)
adam: state(believe(adam,next_subsequence(subsequence(2,[i,s,u,t,w])),true),8,-1)
adam: state(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),
            goal_true),8,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
            barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam:  state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
            blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam:  state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
            blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 9:

```
op(     % do_assert
 if([ conversants(adam,barney),
    act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),goal_true),
   believe(adam,turn(adam),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true) ]),
 effects([ del(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),
           goal_true)) ]),
 atts([ name(do_assert),
   level(information) ]) ).

op(     % request_turn
 if([ conversants(adam,barney),
    act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),goal_true),
   believe(adam,turn(barney),mutually_known_true) ]),
 not([ act(adam,request_turn,true) ]),
 test([ assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney) =
           assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney) ;
           assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney) =
           request(_320,barney) ]),
 action([ act(adam,request_turn,true) ]),
 effects([]),
 atts([ name(request_turn),
   level(turn) ]) ).
```

barney's acts for cycle 9:

```
barney: state(act(adam,request_turn,true),9,-1)
barney: state(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),
          true),9,-1)
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_subsequence(subsequence(2,[blank])),true),5,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(adam,request_turn,true),9,-1)
adam: state(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),
          true),9,-1)
adam: state(believe(adam,turn(adam),mutually_known_true),8,-1)
adam: state(believe(adam,next_subsequence(subsequence(2,[i,s,u,t,w])),true),8,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 10:

```
op(    % asserted_next_subsequence
 if([ conversants(adam,barney),
     believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,blank,g,o]),
           goal_mutually_known_true),
    believe(adam,next_subsequence(subsequence(2,[i,s,u,t,w])),true),
    act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true) ]),
 not([]),
 test([ length([i,s,u,t,w],5),
     length([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,blank,g,o],17),
    2 + 5 =< 17,
     get_next_subsequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,blank,g,o],
           subsequence(2,[i,s,u,t,w]),subsequence(7,[r,q,g,l,d])) ]),
 action([]),
 effects([ del(believe(adam,next_subsequence(subsequence(2,[i,s,u,t,w])),true)),
    del(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true)),
    add(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true)) ]),
 atts([ level(domain),
    name(asserted_next_subsequence) ]) ).
```

barney's acts for cycle 10:

```
op(    % informed_of_next_subsequence_by_other
 if([ conversants(barney,adam),
     believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
           goal_mutually_known_true),
    believe(barney,next_subsequence(subsequence(2,[blank])),true),
    act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true) ]),
 not([]),
 test([ length([blank],1),
     length([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o],17),
    2 + 1 =< 17,
     get_next_subsequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o],
           subsequence(2,[blank]),subsequence(3,[s,u,t,w,r])) ]),
 action([]),
 effects([ del(believe(barney,next_subsequence(subsequence(2,[blank])),true)),
    del(act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true)),
    add(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true)) ]),
 atts([ level(domain),
    name(informed_of_next_subsequence_by_other) ]) ).
```

```
op(     % assertion_received_1
 if([ conversants(barney,adam),
     act(adam,assert(believe(adam,subsequence(2,[i,s,u,t,w]),true),barney),true),
    believe(barney,turn(adam),mutually_known_true) ]),
 not([ believe(adam,subsequence(2,[i,s,u,t,w]),true) ]),
 test([]),
 action([]),
 effects([ add(believe(adam,subsequence(2,[i,s,u,t,w]),true)) ]),
 atts([ name(assertion_received_1),
    level(information) ]) ).
```

```
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(adam),mutually_known_true),8,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 11:

```
op(    % goal_confirm_next_subsequence
 if([ conversants(adam,barney),
     act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,blank,
           blank,g,o])),goal_true),
    believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true) ]),
 not([ act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),
   act(adam,assert(believe(adam,subsequence(7,_307),true),barney),goal_true),
   act(adam,assert(believe(adam,subsequence(7,_323),true),barney),true) ]),
 test([ not(r = blank) ]),
 action([]),
 effects([ add(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true)),
   add(believe(adam,next_letter(letter(1,r)),true)) ]),
 atts([ level(domain),
   name(goal_confirm_next_subsequence) ]) ).
```

barney's acts for cycle 11:

```
op(    % goal_confirm_next_subsequence
 if([ conversants(barney,adam),
     act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,
           g,o])),goal_true),
    believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true) ]),
 not([ act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),
   act(barney,assert(believe(barney,subsequence(3,_2558),true),adam),goal_true),
   act(barney,assert(believe(barney,subsequence(3,_2574),true),adam),true) ]),
 test([ not(s = blank) ]),
 action([]),
 effects([ add(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true)),
   add(believe(barney,next_letter(letter(1,s)),true)) ]),
 atts([ level(domain),
   name(goal_confirm_next_subsequence) ]) ).
```

```
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(adam),mutually_known_true),8,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 12:

```
op(    % goal_confirm_next_letter_1
 if([ conversants(adam,barney),
    act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),
    believe(adam,next_letter(letter(1,r)),true) ]),
 not([ act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true) ]),
 test([]),
 action([]),
 effects([ add(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true)),
    add(act(adam,assert(next_letter(letter(1,r)),barney),goal_true)) ]),
 atts([ level(domain),
    name(goal_confirm_next_letter_1) ]) ).
```

barney's acts for cycle 12:

```
op(    % goal_confirm_next_letter_1
 if([ conversants(barney,adam),
    act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),
    believe(barney,next_letter(letter(1,s)),true) ]),
 not([ act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true) ]),
 test([]),
 action([]),
 effects([ add(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true)),
    add(act(barney,assert(next_letter(letter(1,s)),adam),goal_true)) ]),
 atts([ level(domain),
    name(goal_confirm_next_letter_1) ]) ).
```

```
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),goal_true),12,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(adam,assert(next_letter(letter(1,r)),barney),goal_true),12,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(adam),mutually_known_true),8,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 13:

```
op(    % do_assert
 if([ conversants(adam,barney),
   act(adam,assert(next_letter(letter(1,r)),barney),goal_true),
   believe(adam,turn(adam),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(adam,assert(next_letter(letter(1,r)),barney),true) ]),
 effects([ del(act(adam,assert(next_letter(letter(1,r)),barney),goal_true)) ]),
 atts([ name(do_assert),
   level(information) ]) ).
```

barney's acts for cycle 13:

```
op(    % goal_confirm_next_letter_1
 if([ conversants(barney,adam),
   act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),
   believe(barney,next_letter(letter(1,o)),true) ]),
 not([ act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true) ]),
 test([]),
 action([]),
 effects([ add(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true)),
   add(act(barney,assert(next_letter(letter(1,o)),adam),goal_true)) ]),
 atts([ level(domain),
   name(goal_confirm_next_letter_1) ]) ).

op(    % request_turn
 if([ conversants(barney,adam),
   act(barney,assert(next_letter(letter(1,s)),adam),goal_true),
   believe(barney,turn(adam),mutually_known_true) ]),
 not([ act(barney,request_turn,true) ]),
 test([ assert(next_letter(letter(1,s)),adam) = assert(next_letter(letter(1,s)),adam) ;
         assert(next_letter(letter(1,s)),adam) = request(_2442,adam) ]),
 action([ act(barney,request_turn,true) ]),
 effects([]),
 atts([ name(request_turn),
   level(turn) ]) ).
```

```
barney: state(act(barney,request_turn,true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),goal_true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),13,-1)
barney: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),goal_true),12,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(believe(barney,turn(adam),true),7,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,request_turn,true),13,-1)
adam: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(adam),mutually_known_true),8,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 14:

```
op(     % give_turn_2
 if([ conversants(adam,barney),
   act(barney,request_turn,true),
   believe(adam,turn(adam),mutually_known_true) ]),
 not([ act(adam,assert(next_letter(letter(1,r)),barney),goal_true) ]),
 test([]),
 action([ act(adam,give_turn(barney),true) ]),
 effects([ del(believe(adam,turn(adam),mutually_known_true)),
   del(act(adam,give_turn(barney),true)),
   del(act(barney,request_turn,true)),
   add(believe(adam,turn(barney),mutually_known_true)) ]),
 atts([ name(give_turn_2),
   level(turn) ]) ).
```

barney's acts for cycle 14:

```
op(     % regain_turn
 if([ conversants(barney,adam),
   act(barney,assert(next_letter(letter(1,o)),adam),goal_true),
   believe(barney,turn(adam),true) ]),
 not([ act(barney,hold_turn,mutually_known_true) ]),
 test([ assert(next_letter(letter(1,o)),adam) =
           assert(next_letter(letter(1,o)),adam);
           assert(next_letter(letter(1,o)),adam)
           = request(_2839,adam) ]),
 action([ act(barney,hold_turn,mutually_known_true) ]),
 effects([ del(believe(barney,turn(adam),true)),
   add(believe(barney,turn(barney),true)) ]),
 atts([ name(regain_turn),
   level(turn) ]) ).
```

```
barney: state(believe(barney,turn(barney),true),14,-1)
barney: state(act(barney,hold_turn,mutually_known_true),14,-1)
barney: state(act(adam,give_turn(barney),true),14,-1)
barney: state(act(barney,request_turn,true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),goal_true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),13,-1)
barney: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),goal_true),12,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),7,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,hold_turn,mutually_known_true),14,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),14,-1)
adam: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 15:

barney's acts for cycle 15:

```
op(     % acknowledge_my_turn
 if([ conversants(barney,adam),
   believe(barney,turn(adam),mutually_known_true),
   act(adam,give_turn(barney),true) ]),
 not([]),
 test([]),
 action([ act(barney,acknowledge_turn(turn(barney),mutually_known_true),true) ]),
 effects([ del(act(adam,give_turn(barney),true)),
   del(believe(barney,turn(adam),mutually_known_true)),
   add(believe(barney,turn(barney),mutually_known_true)) ]),
 atts([ level(turn),
   name(acknowledge_my_turn) ]) ).
```

```
barney: state(believe(barney,turn(barney),mutually_known_true),15,-1)
barney: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),true),15,-
        1)
barney: state(believe(barney,turn(barney),true),14,-1)
barney: state(act(barney,hold_turn,mutually_known_true),14,-1)
barney: state(act(barney,request_turn,true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),goal_true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),13,-1)
barney: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),goal_true),12,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
        goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
        goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
        blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),true),15,-1)
adam: state(act(barney,hold_turn,mutually_known_true),14,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),14,-1)
adam: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
        barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
        blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
        blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 16:

barney's acts for cycle 16:

```
op(     % do_assert
 if([ conversants(barney,adam),
   act(barney,assert(next_letter(letter(1,o)),adam),goal_true),
   believe(barney,turn(barney),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(barney,assert(next_letter(letter(1,o)),adam),true) ]),
 effects([ del(act(barney,assert(next_letter(letter(1,o)),adam),goal_true)) ]),
 atts([ name(do_assert),
   level(information) ]) ).
```

```
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),true),16,-1)
barney: state(believe(barney,turn(barney),mutually_known_true),15,-1)
barney: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),
        true),15,-1)
barney: state(believe(barney,turn(barney),true),14,-1)
barney: state(act(barney,hold_turn,mutually_known_true),14,-1)
barney: state(act(barney,request_turn,true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),13,-1)
barney: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),goal_true),12,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(adam,request_turn,true),9,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
        goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
        goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
        blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,assert(next_letter(letter(1,o)),adam),true),16,-1)
adam: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),true),15,-1)
adam: state(act(barney,hold_turn,mutually_known_true),14,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),14,-1)
adam: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
        barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
        blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
        blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 17:

barney's acts for cycle 17:

```
op(     % do_assert
 if([ conversants(barney,adam),
   act(barney,assert(next_letter(letter(1,s)),adam),goal_true),
   believe(barney,turn(barney),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(barney,assert(next_letter(letter(1,s)),adam),true) ]),
 effects([ del(act(barney,assert(next_letter(letter(1,s)),adam),goal_true)) ]),
 atts([ name(do_assert),
   level(information) ]) ).

op(     % give_turn_2
 if([ conversants(barney,adam),
   act(adam,request_turn,true),
   believe(barney,turn(barney),mutually_known_true) ]),
 not([ act(barney,assert(next_letter(letter(1,o)),adam),goal_true) ]),
 test([]),
 action([ act(barney,give_turn(adam),true) ]),
 effects([ del(believe(barney,turn(barney),mutually_known_true)),
   del(act(barney,give_turn(adam),true)),
   del(act(adam,request_turn,true)),
   add(believe(barney,turn(adam),mutually_known_true)) ]),
 atts([ name(give_turn_2),
   level(turn) ]) ).
```

```
barney: state(believe(barney,turn(adam),mutually_known_true),17,-1)
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),true),17,-1)
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),true),16,-1)
barney: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),
          true),15,-1)
barney: state(believe(barney,turn(barney),true),14,-1)
barney: state(act(barney,hold_turn,mutually_known_true),14,-1)
barney: state(act(barney,request_turn,true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),13,-1)
barney: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(act(barney,give_turn(adam),true),17,-1)
adam: state(act(barney,assert(next_letter(letter(1,s)),adam),true),17,-1)
adam: state(act(barney,assert(next_letter(letter(1,o)),adam),true),16,-1)
adam: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),true),15,-1)
adam: state(act(barney,hold_turn,mutually_known_true),14,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),14,-1)
adam: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

adam's acts for cycle 18:

```
op(     % acknowledge_my_turn
 if([ conversants(adam,barney),
   believe(adam,turn(barney),mutually_known_true),
   act(barney,give_turn(adam),true) ]),
 not([]),
 test([]),
 action([ act(adam,acknowledge_turn(turn(adam),mutually_known_true),true) ]),
 effects([ del(act(barney,give_turn(adam),true)),
   del(believe(adam,turn(barney),mutually_known_true)),
   add(believe(adam,turn(adam),mutually_known_true)) ]),
 atts([ level(turn),
   name(acknowledge_my_turn) ]) ).
```

barney's acts for cycle 18:

```
barney: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),18,-1)
barney: state(believe(barney,turn(adam),mutually_known_true),17,-1)
barney: state(act(barney,assert(next_letter(letter(1,s)),adam),true),17,-1)
barney: state(act(barney,assert(next_letter(letter(1,o)),adam),true),16,-1)
barney: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),
          true),15,-1)
barney: state(believe(barney,turn(barney),true),14,-1)
barney: state(act(barney,hold_turn,mutually_known_true),14,-1)
barney: state(act(barney,request_turn,true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,o))),goal_true),13,-1)
barney: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
barney: state(act(barney,confirm_mutual(next_letter(letter(1,s))),goal_true),12,-1)
barney: state(believe(barney,next_letter(letter(1,s)),true),11,-1)
barney: state(act(barney,confirm_mutual(subsequence(3,[s,u,t,w,r])),goal_true),11,-1)
barney: state(believe(adam,subsequence(2,[i,s,u,t,w]),true),10,-1)
barney: state(believe(barney,next_subsequence(subsequence(3,[s,u,t,w,r])),true),10,-1)
barney: state(act(barney,request(act(adam,assert(subsequence(2,[blank]),barney),
          goal_true),adam),true),7,-1)
barney: state(believe(barney,next_letter(letter(1,o)),true),1,-1)
barney: state(believe(barney,sequence([o,blank,s,u,t,w,r,q,g,blank,blank,f,w,w,d,g,o]),
          goal_mutually_known_true),0,-1)
barney: state(act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,g,blank,
          blank,f,w,w,d,g,o])),goal_true),0,-1)
barney: state(conversants(barney,adam),0,-1)


adam: state(believe(adam,turn(adam),mutually_known_true),18,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),18,-1)
adam: state(act(barney,assert(next_letter(letter(1,s)),adam),true),17,-1)
adam: state(act(barney,assert(next_letter(letter(1,o)),adam),true),16,-1)
adam: state(act(barney,acknowledge_turn(turn(barney),mutually_known_true),true),15,-1)
adam: state(act(barney,hold_turn,mutually_known_true),14,-1)
adam: state(act(adam,assert(next_letter(letter(1,r)),barney),true),13,-1)
adam: state(act(adam,confirm_mutual(next_letter(letter(1,r))),goal_true),12,-1)
adam: state(believe(adam,next_letter(letter(1,r)),true),11,-1)
adam: state(act(adam,confirm_mutual(subsequence(7,[r,q,g,l,d])),goal_true),11,-1)
adam: state(believe(adam,next_subsequence(subsequence(7,[r,q,g,l,d])),true),10,-1)
adam: state(act(adam,request_turn,true),9,-1)
adam: state(believe(adam,turn(barney),mutually_known_true),2,-1)
adam: state(act(adam,request(act(barney,assert(subsequence(1,[blank]),adam),goal_true),
          barney),true),2,-1)
adam: state(act(adam,acknowledge_turn(turn(adam),mutually_known_true),true),1,-1)
adam: state(believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o]),goal_mutually_known_true),0,-1)
adam: state(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,blank,
          blank,blank,g,o])),goal_true),0,-1)
adam: state(conversants(adam,barney),0,-1)
```

```
adam's acts for cycle 19:

barney's acts for cycle 19:
```

APPENDIX G

RULE-BASED SYSTEM: PROGRAM

```
% Metalocutionary rule-based system -- program
% David Novick
% October, 1988

ml_version(30).

% The state of a conversant's memory is represented by a set of state
%  relations indexed by the conversant's name in Prolog's internal
%  database.  These relations are of the form:
%
%    state(Act_or_belief,Cycle_added,Cycle_deleted)
%
% The contents of the person's memory consists of a set of acts and beliefs which
%  the person believes have some truth state.  These relations are of the form:
%
%    <act-name>(Actor,Act,Belief_state).
%
%  When an agent performs a conversational act, the act is posted to the active
%  memory of both conversants.
%
%


%%%%%
%%%%%  Program: major cycle
%%%%%

run :-                                                      % Execute multiple cycles
  open(full_trace,append,Stream_1),               % Begin by opening output streams
  open(short_trace,append,Stream_2),                          % First arg is filename
  repeat,                                    % If any operator fires then do_cycle will fail
  do_cycle(Stream_1,Stream_2),                        % thus causing do_cycle to repeat
  close(Stream_1),                                      % If do_cycle succeeds, then done
  close(Stream_2).                                        % so close the output streams


do_cycle(Stream_1,Stream_2) :-        % Perform one conversational cycle for both agents
  increment_cycle,
  portray_active_memory(Stream_1),                                % Stream_1 is full trace
  person_this_cycle(adam,A_acts),                             % Run the cycle for one agent
  display_cycle_result(adam,A_acts,Stream_1,Stream_2),              % and output results
  person_this_cycle(barney,B_acts),                   % Run the cycle for the other agent
  display_cycle_result(barney,B_acts,Stream_1,Stream_2), !,       % Don't retry any of this
  A_acts = [],                                          % Succeed (thus ending repeat)
  B_acts = [].                                                       %  else force repeat
```

```
run_to_cycle(Cycle) :-                        % I.e., run through given cycle and stop
  open(full_trace,append,Stream_1),               % Begin by opening output streams
  open(short_trace,append,Stream_2),                        % First arg is filename
  repeat,                                  % If any operator fires then do_cycle will fail
  do_to_cycle(Cycle,Stream_1,Stream_2),                 % Actually do run the system
  close(Stream_1),                                   % Then close the output streams
  close(Stream_2).                                                     % when done


do_to_cycle(Cycle,Stream_1,Stream_2) :-   % Cycle arg is loop-test arg, not current cycle
  increment_cycle,                                             % Increment loop index
  portray_active_memory(Stream_1),                           %  Stream_1 is full trace
  person_this_cycle(adam,A_acts),                       % Run the cycle for one agent
  display_cycle_result(adam,A_acts,Stream_1,Stream_2),           % and output results
  person_this_cycle(barney,B_acts),                       % Run cycle for other agent
  display_cycle_result(barney,B_acts,Stream_1,Stream_2), !,         % Don't retry above
  cycle(Cycle).                    % Succeed (and thus quit) if end-test val is current cycle


run_single_cycle :-                            % Run system for one cycle for both agents
  open(full_trace,append,Stream_1),                % Begin by opening output streams
  open(short_trace,append,Stream_2),                         % First arg is filename
  do_single_cycle(Stream_1,Stream_2),        % Actually run the system; no repeat needed
  close(Stream_1),                                  % Then close the output strreams
  close(Stream_2).


do_single_cycle(Stream_1,Stream_2) :- !,             % Actually do mechanics of a cycle
  increment_cycle,
  portray_active_memory(Stream_1),                            % Stream_1 is full trace
  person_this_cycle(adam,A_acts),                       % Run the cycle for one agent
  display_cycle_result(adam,A_acts,Stream_1,Stream_2),           % and output results
  person_this_cycle(barney,B_acts),                       % Run cycle for other agent
  display_cycle_result(barney,B_acts,Stream_1,Stream_2).         % and output results


display_cycle_result(Person,Ops,Stream_1,Stream_2) :- !,     % Output results of a cycle
  display_short_result(Person,Ops,user_output),           % Send short trace to terminal
  display_full_result(Person,Ops,Stream_1),          % Send full trace to appropriate file
  display_short_result(Person,Ops,Stream_2).        % Send short trace to appropriate file


display_short_result(Person,Ops,Stream) :- !,         % Output summary of cycle results
  cycle(Cycle),                                                  % What cycle is this?
  set_output(Stream),                               % Send output to appropriate stream
  format("~a's acts for cycle ~d:~n",[Person,Cycle]),                   % Print header
  list_ops_actions(Ops),                  % Print the names of operators fired (if any)
  set_output(user_output), nl.                          % Send output back to terminal
```

```
display_full_result(Person,Ops,Stream) :- !,                    % Output full trace of a cycle
  cycle(Cycle),                                                       % What cycle is this?
  set_output(Stream),                                % Send output to appropriate stream
  format("~2n~a's acts for cycle ~d:~n",[Person,Cycle]),                   % Print header
  portray_ops(Ops),                        % Print full account of operators fired (if any)
  set_output(user_output), nl.                          % Send the output back to terminal


list_ops_actions([]) :- !.                         % If no more operators to print then done

list_ops_actions([Op|Ops]) :- !,          % Print the name and actions of the first op of a list
  get_attribute(name(Name),Op),                                  % Get the operator's name
  format("  ~a --~n",[Name]),                                                    % Print it
  list_actions(Op),                                % Print any actions taken by the operator
  list_ops_actions(Ops).                           % Do the same thing for the rest of the list


list_actions(op(_,_,_,action(Actions),_,_)) :-      % Print any actions taken by an operator
  list_actions_1(Actions).                                               % Simplify predicate


list_actions_1([]) :- !.                                      % If no more actions then done

list_actions_1([A|As]) :-                                     % Print a list of operator actions
  format("     ~w~n",[A]),                                              % Print the first action
  list_actions_1(As).                                      % And then print the rest of the list


portray_ops([]) :- !.                           % If no more operators to print, then done

portray_ops([Op|Ops]) :- !,               % Print the full Prolog form of a list of operators
  portray_op(Op),                                                  % Print the first operator
  portray_ops(Ops).                                         % And the print the rest of the list


portray_op(op(if(I),not(N),test(T),action(A),effects(E),atts(Atts))) :- !,    % In Prolog form
  member(name(Name),Atts),                           % Get the name of the operator from atts list
  format("~nop(    % ~a",[Name]),       % Print term-name and (as comment) operator name
  portray_op_clause(if,I),          % Then print the various sub-terms which make up the op
  portray_op_clause(not,N),
  portray_op_clause(test,T),
  portray_op_clause(action,A),
  portray_op_clause(effects,E),
  portray_last_op_clause(atts,Atts),              % atts term is special because it is the last one
  format(" ).~n",[]).                                          % Finally, close off the op term
```

```
portray_op_clause(Label,[]) :- !,                          % If arg list is empty
  format("~n  ~a([]),",[Label]).            %  then just print term with empty arg list

portray_op_clause(Label,[Term]) :- !,          % If term has arg list with one term in it
  format("~n  ~a([ ~w ]),",[Label,Term]).                    % then print it accordingly

portray_op_clause(Label,[Term|Terms]) :- !,    % If term has arg list with multiple terms
  format("~n  ~a([ ~w,",[Label,Term]),           % then print term name and first arg,
  portray_other_op_clause_terms(Terms),             % then print the rest of the args,
  format(" ]),",[]).                                % then close the list and the term


portray_last_op_clause(Label,[]) :- !,         % This predicate is like portray_op_clause
  format("~n  ~a([])",[Label]).              % except that it leaves off the trailing comma

portray_last_op_clause(Label,[Term]) :- !,                % because the op term is ending
  format("~n  ~a([ ~w ])",[Label,Term]).

portray_last_op_clause(Label,[Term|Terms]) :- !,
  format("~n  ~a([ ~w,",[Label,Term]),
  portray_other_op_clause_terms(Terms),
  format(" ])",[]).


portray_other_op_clause_terms([]) :- !.                % No more terms to print; so done

portray_other_op_clause_terms([Term]) :- !,                    % Print the last term in a list
  format("~n     ~w",[Term]).

portray_other_op_clause_terms([Term|Terms]) :- !,                  % Print the tail of a list
  format("~n     ~w,",[Term]),                 % by printing the first term of the tail
portray_other_op_clause_terms(Terms).                      % and then the rest of the terms
```

```
%%
%% Clauses which perform the match-resolve-execute cycle
%%

increment_cycle :-              % Change the program to increment the current cycle number
  retract(cycle(C)),
  D is C + 1,
  assert(cycle(D)).


person_this_cycle(Person,Selections) :- !,       % Mechanics of a single cycle for one agent
  match_ops(Person,Matches),              % Match the operators against the knowledge base
  select_ops(Matches,Selections),         % Resolve conflicts among the matching operators
  do_ops(Person,Selections).                     % Then execute the selected operators


%
% Clauses which perform the "execute" functions of the cycle
%

do_ops(_,[]) :- !.                                               % No ops to fire, so done

do_ops(Person,[op(_,_,_,action(A),effects(E),_)|Ops]) :- !,        % Fire a list of operators
  do_acts(Person,A),                                        % For the first op, do the acts
  do_effects(Person,E),                                                     % do the effects
  do_ops(Person,Ops).                                        % Then fire the rest of the ops


do_acts(Person,[]) :- !.                                         % No acts to do, so done

do_acts(Person,[Act|Acts]) :-                                          % To  do a list of acts
  do_act(Person,Act),                                                      % Do the first act
  do_acts(Person,Acts).                                    % Then do the rest of the acts


do_act(Person,Act) :- !,                                         Do an act; no need to retry
  cycle(Cycle),                                                % What is the current cycle?
  recorded(Person,state(conversants(Person,Other),0,-1),_),    % Who are relevant agents?
  recorda(Person,state(Act,Cycle,-1),_),         % Add the act to each agent's knowledge base
  recorda(Other,state(Act,Cycle,-1),_).       % indicating added current cycle and not deleted


do_effects(_,[]) :- !.                                        % No effects to do, so done

do_effects(Person,[Effect|Effects]) :-                              % To do a list of effects
  do_effect(Person,Effect),                                              % Do the first effect
  do_effects(Person,Effects).                             % Then do the rest of the effects
```

```
do_effect(_,[]).                                              % Base case for completeness

do_effect(Person,add(State)) :-                  % Add a state to an agent's knowledge base
 cycle(Cycle),                                                % What is the current cycle?
 recorda(Person,state(State,Cycle,-1),_).   % Indicate added current cycle and not deleted

do_effect(Person,del(State)) :-             % Delete a state from an agent's knowledge base
 cycle(Cycle),                                                % What is the current cycle?
 recorded(Person,state(State,Cycle_added,_),Ref),      % Remember the fact to be deleted
 erase(Ref),                                                            % Really delete it
 recorda(Person,state(State,Cycle_added,Cycle),_).  % Re-record, showing deletion cycle


add_state(Person,State) :-                          % Used for initialization and debugging
 del_if_exists(Person1,State),                                 % Delete possible duplicate
 add(Person,State).                                   % Add the state on the current cycle


del_if_exists(Person1,act(Person2,Act,Truth_value)) :-                        % Delete an act
 del(Person1,act(Person2,Act,_)).

del_if_exists(Person1,believe(Person2,Belief,Truth_value)) :-              % Delete a belief
 del(Person1,believe(Person2,Act,_)).

del_if_exists(_,_).                                   % If nothing matches, succeed anyway


del(Person,State) :-                                % Used for initialization and debugging
 cycle(Cycle),                                                % What is the current cycle?
 recorded(Person,state(State,Cycle_added,_),Ref),      % Remember the fact to be deleted
 erase(Ref),                                                            % Really delete it
 recorda(Person,state(State,Cycle_added,Cycle),_).  % Re-record, showing deletion cycle


add(Person,State) :-                                % Used for initialization and debugging
 cycle(Cycle),                                                % What is the current cycle?
 recorda(Person,state(State,Cycle,-1),_).             % Add the state on the current cycle


%
% Clauses which perform the "resolve-conflicts" part of the cycle
%

select_ops([],[]) :- !.                                  % No ops matched, so nothing to do
select_ops(Possible_ops,Chosen_ops) :-                   % To resolve conflicts among ops
 check_for_opportunities(Possible_ops,Opp_ops),         % Try to coordinate inter-level ops
 check_consistent_effects(Opp_ops,Chosen_ops).               % Make sure ops don't conflict
```

```
check_for_opportunities(P,P).                                          % Stubbed


check_consistent_effects(Ops1,Ops3) :-                  % To make sure ops don't conflict
  unique_levels(Ops1,Ops2),                   % Discard multiple ops on any particular level
  lowest_level_op(Ops2,Lowest,Others),           % Find the lowest-level op and key off it
  most_consistent_effects([Lowest|Others],Ops3).    % Discard ops whose effects conflict


unique_levels(Ops1,Ops2) :- !,                % Discard multiple ops on any particular level
  unique_levels_1(Ops1,Ops2,[]).        % Set up predicate with empty list for level names


unique_levels_1([],[],_).                                % No more ops to look at, so done

unique_levels_1([Op|Ops],Ops2,Levels) :-   % To drop an op if already have same level op
  get_attribute(level(Level),Op),                            % Get level of first op in list
  member(Level,Levels),                               % Do we already have an op at this level?
  unique_levels_1(Ops,Ops2,Levels).                   % If so, don't add this op to list returned

unique_levels_1([Op|Ops],[Op|Ops2],Levels) :-                  % Else op's level must be new
  get_attribute(level(Level),Op),                                      % So get the op's level
  unique_levels_1(Ops,Ops2,[Level|Levels]).   % And add the new level to the list of levels


lowest_level_op([Op],Op,[]) :- !.                          % Only 1 op left, so must be lowest

lowest_level_op([Op1,Op2|Ops1],Op3,[Op2|Ops2]) :-                % To find lowest level op
  lower_level_op(Op1,Op2),                              % check order of first two ops in list
  lowest_level_op(Ops1,Op3,Ops2).                       % If lower, skip 2nd op & check rest
lowest_level_op([Op1,Op2|Ops1],Op3,[Op1|Ops2]) :-                    % Else 2d op was lower

  lowest_level_op([Op2|Ops1],Op3,Ops2).                          % So skip first op & check rest


lower_level_op(op(_,_,_,_,_,atts(Atts1)),op(_,_,_,_,_,atts(Atts2))) :-     % Compare ops
  member(level(D1),Atts1),                                                 % Get level of first op
  member(level(D2),Atts2),                                               % Get level of second op
  level_order(D2,D1).                                   % See if first level is higher than second

% level_order: as other levels are created in ops, additional clauses
%  need to be added. Actually, a more efficient system will be needed for this,
%  because the number of pairs will go up too fast. This could be changed, for
%  example, to check relative positions in a list.

level_order(domain,turn).
level_order(domain,information).
level_order(information,turn).
```

```
most_consistent_effects([Op|Rest],Consistent_ops) :- !,     % Assuming first op has priority
  most_consistent_effects_1(Op,Rest,Consistent_ops).        % make sure rest don't conflict


most_consistent_effects_1(Op,[],[Op]).                            % Just one op, so must be ok

most_consistent_effects_1(Op1,[Op2|Ops],[Op2|Ok_ops]) :-              % List of ops is ok if
  consistent_effects(Op1,Op2),                                % First op is ok with priority op
  most_consistent_effects_1(Op1,Ops,Ok_ops).               % And the rest of the list is pruned

most_consistent_effects_1(Op,[_|Ops],All_ok_ops) :-                   % If first op is not ok
  most_consistent_effects_1(Op,Ops,All_ok_ops).               % Skip it and prune rest of list


consistent_effects(op(_,_,_,action(A1),effects(E1),_),               % 2 ops are consistent
                   op(_,_,_,action(A2),effects(E2),_)) :-                               % if
  combine_acts_and_effects(A1,E1,C1),               % taking first op's acts & effects together
  combine_acts_and_effects(A2,E2,C2),                % taking 2d op's acts & effects together
  consistent_effects_1(C1,C2).                         % there isn't any conflict among them


combine_acts_and_effects(A,E,C1) :-                 % To combine acts and effects into a list
  actions_format(A,S),                             % put the acts into same format as effects
  append(S,E,C1).                                                % and then just append them


actions_format([],[]) :- !.                                  % No more actions left, so done

actions_format([A|As],[add(A)|Rest]) :-   % An act is always added, so wrap with add code
  actions_format(As,Rest).                                  % Then format the rest of the acts


consistent_effects_1([],_).                            No more effects left to check, so done

consistent_effects_1([E1|R1],E2) :-                           % E2 is list of effects from 2d op
  consistent_effect(E1,E2),          % Check first effect of first op against all effects of 2d op
  consistent_effects_1(R1,E2).                                % Check rest of effects of first op


consistent_effect(_,[]).                                  % No more effects to check, so done

consistent_effect(add(S),[del(S)|_]) :- !, fail.       % Not consistent to add and del same fact

consistent_effect(del(S),[add(S)|_]) :- !, fail.       % Not consistent to add and del same fact

consistent_effect(E1,[_|R2]) :-                              % Else effects are consistent so far
  consistent_effect(E1,R2).                      % so check against rest of list of effects of 2d op
```

```
%
% Clauses which perform the "match" part of the cycle
%

match_ops(Person,Ops) :-                          % For a given person, there is a set of ops
  cycle(Cycle),                                           % which for the current cycle
  bagof(Op,op_premises_true_for_cycle(Person,Op,Cycle),Ops).          % matches in kb

match_ops(_,[]) :- !.                            % Else no matches, so return empty set


op_premises_true_for_cycle(Person,                           % To match a particular op
       op(if(I),not(N),test(T),action(A),effects(E),atts(Atts)),Cycle) :-
  op(if(I),not(N),test(T),action(A),effects(E),atts(Atts)),           % Find an op in memory
  check_ifs(Person,I,Cycle),                                 % Check the op's if clauses
  check_nots(Person,N,Cycle),                               % Check the op's not clauses
  check_tests(T).                                           % Check the op's test clauses


check_ifs(_,[],_) :- !.                          % No more if clauses to check, so done

check_ifs(Person,[If_clause|Rest],Cycle) :-                        % To check if clauses
  recorded(Person,state(If_clause,Cycle_added,Cycle_deleted),Key),            % match kb
  true_for_cycle(Cycle_added,Cycle_deleted,Cycle),     % make sure clause is currently true
  check_ifs(Person,Rest,Cycle).                          % and check the rest of the clauses


true_for_cycle(Cycle_added,Cycle_deleted,Cycle) :-      % A fact is true on a given cycle if
  Cycle_added < Cycle,                                     % it was added before this cycle
  (Cycle_deleted =:= -1 ; Cycle < Cycle_deleted ).            % and hasn't been deleted yet


check_nots(_,[],_) :- !.                      % No more 'not' clauses to check, so done

check_nots(Person,[Not_clause|Rest],Cycle) :-              % A 'not' clause should succeed
  \+ (recorded(Person,state(Not_clause,_,_),Key)),    % if matching fact never posted to kb
  check_nots(Person,Rest,Cycle).                         % Then check rest of 'not' clauses

check_nots(Person,[Not_clause|Rest],Cycle) :-          % If a matching fact had been posted
  recorded(Person,state(Not_clause,Cycle_added,Cycle_deleted),Key), !,
  not(true_for_cycle(Cycle_added,Cycle_deleted,Cycle)),           % 'not' ok if not true now
  check_nots(Person,Rest,Cycle).                              % Then check rest of 'nots'


check_tests([]) :- !.                                % No more tests to check, so done
check_tests([H|L]) :- !, H, check_tests(L).                   % Eval test, then check rest


get_attribute(Att,op(_,_,_,_,_,atts(Atts))) :-                 To get an attribute of an op
  member(Att,Atts).                                       % match against the ops atts list
```

```
%%%%%
%%%%% Development aids
%%%%%

reinitialize :-                              % To set the simulation to its initial state
  reinitialize_ops,                            % make sure the ops are all up to date
  reinitialize_memory.                         % and reinitialize the knowledge base


reinitialize_ops :-                                        % To get all ops up to date
  abolish(op,6),                                   % erase all op clauses in the system
  consult([ml1do,ml1mo]).                          % and read them in from their files


reinitialize_memory :-                                     % To reinitialize memory
  current_key(_,Key),                             % find any key to the knowledge base
  erase_facts_for_key(Key),                          % erase all the facts for that key
  fail.                                          % and force trying of some other key

reinitialize_memory :- !,                 % All keys must have been used, so kb is blank
  consult(ml1si).                                % So read in initial state from its file


erase_facts_for_key(Key) :-                   % To erase all facts in kb for a given key
  recorded(Key,_,Ref),                                            % match agains the kb
  erase(Ref),                                                      % erase that fact
  fail.                                       % and force rematches to erase other facts

erase_facts_for_key(_) :- !.             % Must have erased all facts for this key, so done


memory :-                              % To display the entire knowlege base for all agents
  current_key(_,Key),                   % find any key in the kb (will be an agent's name)
  display_facts_for_key(Key),                        % display all the facts for that agent
  fail.                                     % and force rematch of key for other agents

memory :- !.                                        % Must have matched all keys, so done


display_facts_for_key(Key) :-                       % To display the kb for a given agent
  recorded(Key,Term,Ref),                                   % find any fact for that agent
  format("~n~a: ~w~n {~w}",[Key,Term,Ref]),                                   % print it
  fail.                                     % and force rematches to print rest of facts

display_facts_for_key(_) :- !.            % Must have printed all facts for agent, so done
```

```
active_memory :-                              % To display active facts in kb for all agents
 cycle(Cycle),                                                    % get the current cycle
 current_key(_,Key),                               % find any key in the kb (will be agent)
 display_facts_for_key_and_cycle(Key,Cycle),   % display all the facts for agent this cycle
 fail.                                           % and force rematch of key for other agents

active_memory :- !.                                    Must have matched all keys, so done

display_facts_for_key_and_cycle(Key,Cycle) :-            % To display kb for agent on cyle
 recorded(Key,state(S,Cycle_added,Cycle_deleted),Ref),                    % find any fact
 Cycle_added < Cycle,                                   % that was added before the cycle
 (Cycle_deleted = -1 ;Cycle_deleted >= Cycle),          % and not deleted until after cycle
 format("~n~a: ~w~n {~w}",[Key,state(S,Cycle_added,Cycle_deleted),Ref]),     % print it
 fail.                                       % and force rematches of facts to print rest

display_facts_for_key_and_cycle(_) :- !.             % Must have printed all facts, so done


portray_active_memory(Stream) :-          % To send display of active facts in kb to a stream
 set_output(Stream),                                          % direct output to the steam
 cycle(Cycle),                                                          % get current cycle
 current_key(_,Key), nl, nl,                      % find any agent, print some blank lines
 portray_facts_for_key_and_cycle(Key,Cycle),             % display the facts for that agent
 fail.                                           % and force rematch of key for other agents

portray_active_memory(_) :-                                         Must have matched all keys
 set_output(user_output), !.                               % so redirect output to terminal


portray_facts_for_key_and_cycle(Key,Cycle) :-                    % Like display_facts, but
 recorded(Key,state(S,Cycle_added,Cycle_deleted),_),               % slightly simpler output
 Cycle_added < Cycle,
 (Cycle_deleted = -1 ;Cycle_deleted >= Cycle),
 format("~a: ~w~n",[Key,state(S,Cycle_added,Cycle_deleted)]),
 fail.

portray_facts_for_key_and_cycle(_) :- !.             % Must have matched all facts, so done


back_to_cycle(Cycle) :-               % To return the system to its state at start of some cycle
  erase_facts_added_from(Cycle),           % erase all the facts added on and after that cycle
  undelete_facts_deleted_from(Cycle),      % restore the facts deleted on and after that cycle
  reset_cycle(Cycle),                                    % change the state of the cycle index
  format("{ Cycle is now ~D }",[Cycle]).                        % and tell user we've done it


reset_cycle(New_cycle) :- !,                           % To change the system's cycle index
 retract(cycle(_)),                                                 % erase the current index
 assert(cycle(New_cycle)).                                         % and assert the new value
```

```
erase_facts_added_from(Cycle) :-            % To erase all facts added on and after cycle
  current_key(_,Key),                                              % find any key in the kb
  recorded(Key,state(_,Cycle_added,_),Ref),                   % find any fact with that key
  Cycle_added >= Cycle,                          % check if fact added on or after the cycle
  erase(Ref),                                                     % if so, erase it from kb
  fail.                                             % and force rematch of facts and keys

erase_facts_added_from(_) :- !.           % Must have matched all keys and facts, so done


undelete_facts_deleted_from(Cycle) :-         % To restore facts deleted on and after cycle
  current_key(_,Key),                                                       % find any key
  recorded(Key,state(State,Cycle_added,Cycle_deleted),Ref),  % find any fact with the key
  Cycle_deleted >= Cycle,                         % check if key deleted on or after the cycle
  erase(Ref),                                                      % if so, erase the fact and
  recorda(Key,state(State,Cycle_added,-1),_),   % set del cycle to show fact not yet deleted
  fail.                                             % and force rematch of facts and keys

undelete_facts_deleted_from(_) :- !.      % Must have matched all keys and facts, so done


check_op(Person,Name) :-                       % See if named op will match on this cycle
  cycle(Cycle),                                                             % Get the cycle
  op(if(I),not(N),test(T),_,_,atts(Atts)),                                      % Get an op
  member(name(Name),Atts),                                       % Make sure its the right one
  undo(notrace),                                              % On backtrack turn off trace mode
  trace,                                                                 % Turn on trace mode
  check_ifs(Person,I,Cycle),                  % Look at system checking 'ifs' against memory
  check_nots(Person,N,Cycle),               % Look at system checking 'nots' against memory
  check_tests(T).                              % Look at system checking tests against memory

do_cycle :-                                                            % To run a single cycle
  increment_cycle,                                                  % increment the cycle index
  person_this_cycle(adam,A_acts),                                     % run the cycle for adam
  display_cycle_result(adam,A_acts,user_output),                              % show the results
  person_this_cycle(barney,B_acts),                                  % run the cycle for barney
  display_cycle_result(barney,B_acts,user_output), !,                         % show the results
  A_acts = [],                                              % succeed (thus ending any repeat)
  B_acts = [].                                                 %  when both have nothing to do

display_named_op(Name) :-                                          % To print a named operator
  op(I,N,T,A,E,atts(Atts)),                                                        % get an op
  member(name(Name),Atts),                                       % make sure its the right one
  portray_op(op(I,N,T,A,E,atts(Atts))).                                        % and print it out
```

APPENDIX H

RULE-BASED SYSTEM: META-LOCUTIONARY OPERATORS

```
% Meta-locutionary rule-based system -- meta-locutionary operators
% last rev. 10-21-88


%%%%%
%%%%%  information-level operators
%%%%%
%
%  do_assert
%  assertion_received_1
%  do_request
%  request_received_2
%


op(      % do_assert
 if([    conversants(Me,Other),
         act(Me,assert(Info,Other),goal_true),
         believe(Me,turn(Me),mutually_known_true) ]),
 not([]),
 test([ ]),
 action([ act(Me,assert(Info,Other),true) ]),
 effects([ del(act(Me,assert(Info,Other),goal_true)) ]),
 atts([ name(do_assert),
         level(information) ]) ).


op(      % assertion_received_1
 if([    conversants(Me,Other),
         act(Other,assert(believe(Other,Info,Belief_value),Me),true),
         believe(Me,turn(Other),mutually_known_true) ]),
 not([ believe(Other,Info,Belief_value) ]),
 test([]),
 action([]),
 effects([ add(believe(Other,Info,Belief_value)) ]),
 atts([ name(assertion_received_1),
         level(information) ]) ).
```

```
op(     % do_request
 if([   conversants(Me,Other),
        act(Me,request(Info,Other),goal_true),
        believe(Me,turn(Me),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(Me,request(Info,Other),true) ]),
 effects([ del(act(Me,request(Info,Other),goal_true)),
        del(believe(Me,turn(Me),mutually_known_true)),
        add(believe(Me,turn(Other),mutually_known_true)) ]),
 atts([ name(do_request),
        level(information) ]) ).


op(     % request_received_2
 if([   conversants(Me,Other),
        act(Other,request(act(Me,Act,goal_true),Me),true),
        act(Me,Act,goal_true) ]),
 not([]),
 test([]),
 action([]),
 effects([ del(act(Other,request(act(Me,Act,goal_true),Me),true)) ]),
 atts([ name(request_received_2),
        level(information) ]) ).


%%%%%
%%%%%  turn-level operators
%%%%%
%
%  acknowledge_my_turn
%  my_turn_acknowledged
%  acknowledge_others_turn
%  others_turn_acknowledged
%  recognize_my_turn_1
%  recognize_my_turn_2
%  give_turn_1
%  give_turn_2
%  request_turn
%  hold_turn
%
```

```
op(     % acknowledge_my_turn
 if([   conversants(Me,Other),
     believe(Me,turn(Other),mutually_known_true),
        act(Other,give_turn(Me),true) ]),
 not([]),
 test([]),
 action([ act(Me,acknowledge_turn(turn(Me),mutually_known_true),true) ]),
 effects([ del(act(Other,give_turn(Me),true)),
     del(believe(Me,turn(Other),mutually_known_true)),
        add(believe(Me,turn(Me),mutually_known_true)) ]),
 atts([ level(turn),
        name(acknowledge_my_turn) ]) ).


op(     % my_turn_acknowledged
 if([   conversants(Me,Other),
     believe(Me,turn(Me),true),
        act(Other,acknowledge_turn(turn(Me),mutually_known_true),true) ]),
 not([]),
 test([]),
 action([]),
 effects([ del(believe(Me,turn(Me),true)),
        add(believe(Me,turn(Me),mutually_known_true)),
        del(act(Other,acknowledge_turn(turn(Me),mutually_known_true),true)) ]),
 atts([ level(turn),
        name(my_turn_acknowledged) ]) ).


op(     % acknowledge_others_turn
 if([   conversants(Me,Other),
        believe(Me,turn(Me),mutually_known_true),
        act(Other,take_turn,true) ]),
 not([ act(Me,Act,goal_true) ]),
 test([]),
 action([ act(Me,acknowledge_turn(turn(Other),mutually_known_true),true) ]),
 effects([ del(act(Other,take_turn,true)),
        del(act(Me,acknowledge_turn(turn(Other),mutually_known_true),true)),
        del(believe(Me,turn(Me),mutually_known_true)),
        add(believe(Me,turn(Others),mutually_known_true)) ]),
 atts([ level(turn),
        name(acknowledge_others_turn) ]) ).
```

```
op(     % others_turn_acknowledged
 if([    conversants(Me,Other),
      believe(Me,turn(Other),true),
         act(Other,acknowledge_turn(turn(Other),mutually_known_true),true) ]),
 not([]),
 test([]),
 action([]),
 effects([ del(believe(Me,turn(Other),true)),
   add(believe(Me,turn(Other),mutually_known_true)),
   del(act(Other,acknowledge_turn(turn(Other),mutually_known_true),true)) ]),
 atts([ level(turn),
   name(others_turn_acknowledged) ]) ).


op(     % recognize_my_turn_1
 if([    conversants(Me,Other),
         act(Other,give_turn(Me),mutually_known_true),
         believe(Me,turn(Other),_) ]),
 not([ act(Me,request-turn,true) ]),
 test([]),
 action([]),
 effects([ del(believe(Me,turn(Other),_)),
         del(act(Other,give_turn(Me),mutually_known_true)),
         add(believe(Me,turn(Me),mutually_known_true)) ]),
 atts([ name(recognize_my_turn_1),
         level(turn) ]) ).


op(     % recognize_my_turn_2
 if([    conversants(Me,Other),
         act(Other,give_turn(Me),mutually_known_true),
         act(Me,request-turn,true),
         believe(Me,turn(Other),_) ]),
 not([]),
 test([]),
 action([]),
 effects([ del(believe(Me,turn(Other),_)),
         del(act(Me,request-turn,true)),
         del(act(Other,give_turn(Me),mutually_known_true)),
         add(believe(Me,turn(Me),mutually_known_true)) ]),
 atts([ name(recognize_my_turn_2),
         level(turn) ]) ).
```

```
op(     % give_turn_1
 if([    conversants(Me,Other),
         act(Other,Act,goal_true),
         believe(Me,turn(Me),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(Me,give_turn(Other),true) ]),
 effects([ del(believe(Me,turn(Me),mutually_known_true)),
         del(act(Me,give_turn(Other),true)),
         add(believe(Me,turn(Other),true)) ]),
 atts([ name(give_turn_1),
         level(turn) ]) ).


op(     % give_turn_2
 if([ conversants(Me,Other),
   act(Other,request_turn,true),
   believe(Me,turn(Me),mutually_known_true) ]),
 not([ act(Me,Act,goal_true) ]),
 test([]),
 action([ act(Me,give_turn(Other),true) ]),
 effects([ del(believe(Me,turn(Me),mutually_known_true)),
         del(act(Me,give_turn(Other),true)),
         del(act(Other,request_turn,true)),
         add(believe(Me,turn(Other),mutually_known_true)) ]),
 atts([ name(give_turn_2),
         level(turn) ]) ).


op(     % give_turn_3
 if([    conversants(Me,Other),
         act(Me,request(Info,Other),goal_true),
         believe(Me,turn(Me),mutually_known_true) ]),
 not([]),
 test([]),
 action([ act(Me,give_turn(Other),mutually_known_true) ]),
 effects([ del(believe(Me,turn(Me),mutually_known_true)),
         del(act(Me,give_turn(Other),mutually_known_true)),
         add(believe(Me,turn(Other),true)) ]),
 atts([ name(give_turn_3),
         level(turn) ]) ).
```

```
op(     % request_turn
 if([    conversants(Me,Other),
         act(Me,Act,goal_true),
         believe(Me,turn(Other),mutually_known_true) ]),
 not([ act(Me,request_turn,true) ]),
 test([ ( Act = assert(Info,Other) ; Act = request(Req,Other) ) ]),
 action([ act(Me,request_turn,true) ]),
 effects([]),
 atts([ name(request_turn),
         level(turn) ]) ).


op(     % hold_turn
 if([    conversants(Me,Other),
         act(Me,Act,goal_true),
         believe(Me,turn(Other),true) ]),
 not([ act(Me,hold_turn,mutually_known_true) ]),
 test([ ( Act = assert(Info,Other) ; Act = request(Req,Other) ) ]),
 action([ act(Me,hold_turn,mutually_known_true) ]),
 effects([ del(believe(Me,turn(Other),true)),
         add(believe(Me,turn(Me),true)) ]),
 atts([ name(regain_turn),
         level(turn) ]) ).
```

APPENDIX I

RULE-BASED SYSTEM:
DOMAIN OPERATORS

```
% Meta-locutionary rule-based system -- domain operators
% last rev. 10-21-88

%%%%%
%%%%%  Domain-level operators
%%%%%
%
%       goal_confirm_next_subsequence
%       confirmed_next_subsequence
%       confirmed_last_subsequence
%       goal_confirm_next_letter_1
%       goal_confirm_next_letter_2
%       confirmed_next_letter
%       confirmed_last_letter
%       goal_request_next_subsequence
%       obtained_next_subsequence
%       obtained_last_subsequence
%       obtained_next_letter
%       obtained_last_letter
%       goal_assert_next_subsequence
%       asserted_next_subsequence
%       asserted_last_subsequence
%       informed_of_next_subsequence_by_other
%       informed_of_last_subsequence_by_other
%


op(     % goal_confirm_next_subsequence
 if([   conversants(Me,Other),
        act(Me,confirm_mutual(sequence(Sequence)),goal_true),
        believe(Me,next_subsequence(subsequence(Index,[L|R])),true) ]),
 not([ act(Me,confirm_mutual(subsequence(Index,[L|R])),goal_true),
        act(Me,assert(believe(Me,subsequence(Index,_),true),Other),goal_true),
        act(Me,assert(believe(Me,subsequence(Index,_),true),Other),true) ]),
 test([ not(L = blank) ]),
 action([]),
 effects([ add(act(Me,confirm_mutual(subsequence(Index,[L|R])),goal_true)),
        add(believe(Me,next_letter(letter(1,L)),true)) ]),
 atts([ level(domain),
        name(goal_confirm_next_subsequence) ]) ).
```

```
op(     % confirmed_next_subsequence
 if([ conversants(Me,Other),
        believe(Me,sequence(Sequence),goal_mutually_known_true),
        believe(Me,next_subsequence(subsequence(Index,List)),true),
        act(Me,confirm_mutual(subsequence(Index,List)),goal_true),
        act(Me,confirm_mutual(next_letter(letter(L_index,Char))),true) ]),
 not([]),
 test([ length(List,Length),
        L_index >= Length,
        get_next_subsequence(Sequence,
               subsequence(Index,List),
               subsequence(New_index,New_list)) ]),
 action([ act(Me,confirm_mutual(subsequence(Index,List)),true) ]),
 effects([ del(act(Me,confirm_mutual(subsequence(Index,List)),goal_true)),
        del(act(confirm_mutual(next_letter(letter(L_index,Char))),true)),
        add(believe(Me,next_subsequence(New_index,New_list),true)) ]),
 atts([ level(domain),
        name(confirmed_next_subsequence) ]) ).


op(     % confirmed_last_subsequence
 if([ conversants(Me,Other),
        believe(Me,sequence(Sequence),goal_mutually_known_true),
        believe(Me,next_subsequence(subsequence(Index,List)),true),
        act(Me,confirm_mutual(subsequence(Index,List)),goal_true),
        act(Me,confirm_mutual(next_letter(letter(L_index,Char))),true) ]),
 not([]),
 test([ length(List,Len1),
        length(Sequence,Len2),
        L_index >= Len1,
        Index + Len1 > Len2 ]),
 action([ act(Me,confirm_mutual(subsequence(Index,List)),true) ]),
 effects([ del(act(Me,confirm_mutual(subsequence(Index,List)),goal_true)),
        del(act(Me,confirm_mutual(next_letter(letter(L_index,Char))),true)) ]),
 atts([ level(domain),
        name(confirmed_last_subsequence) ]) ).


op(     %  goal_confirm_next_letter_1
 if([ conversants(Me,Other),
   act(Me,confirm_mutual(subsequence(_,_)),goal_true),
   believe(Me,next_letter(Letter),true) ]),
 not([ act(Me,confirm_mutual(next_letter(Letter)),goal_true) ]),
 test([]),
 action([]),
 effects([ add(act(Me,confirm_mutual(next_letter(Letter)),goal_true)),
   add(act(Me,assert(next_letter(Letter),Other),goal_true)) ]),
 atts([ level(domain),
   name(goal_confirm_next_letter_1) ]) ).
```

```
op(     %  goal_confirm_next_letter_2
 if([ conversants(Me,Other),
   act(Me,confirm_mutual(subsequence(_,_)),goal_true),
   believe(Me,next_letter(Letter),true),
   act(Me,confirm_mutual(next_letter(Other_letter)),goal_true) ]),
 not([ act(Me,confirm_mutual(next_letter(Letter)),goal_true) ]),
 test([ not(Letter = Other_letter) ]),
 action([]),
 effects([ add(act(Me,confirm_mutual(next_letter(Letter)),goal_true)),
   del(act(Me,confirm_mutual(next_letter(Other_letter)),goal_true)),
   add(act(Me,assert(next_letter(Letter),Other),goal_true)) ]),
 atts([ level(domain),
   name(goal_confirm_next_letter_2) ]) ).


op(     % confirmed_next_letter
 if([ conversants(Me,Other),
   act(Me,confirm_mutual(subsequence(S_index,List)),goal_true),
   act(Me,confirm_mutual(next_letter(letter(Index,Letter))),goal_true),
   believe(Me,next_letter(letter(Index,Letter)),mutually_known_true) ]),
 not([]),
 test([ get_next_letter(Index,List,New_letter) ]),
 actions([]),
 effects([
   del(act(Me,confirm_mutual(next_letter(letter(Index,Letter))),goal_true)),
   add(act(Me,confirm_mutual(next_letter(letter(Index,Letter))),true)),
   del(believe(Me,next_letter(letter(Index,Letter)),mutually_known_true)),
   add(believe(Me,next_letter(New_letter),true)) ]),
 atts([ name(confirmed_next_letter),
   level(domain) ]) ).


op(     % confirmed_last_letter
 if([ conversants(Me,Other),
   act(Me,confirm_mutual(subsequence(S_index,List)),goal_true),
   act(Me,confirm_mutual(next_letter(letter(Index,Letter))),goal_true),
   believe(Me,next_letter(letter(Index,Letter)),mutually_known_true) ]),
 not([]),
 test([ length(List,Length),
        Index >= Length ]),
 action([]),
 effects([
   del(act(Me,confirm_mutual(next_letter(letter(Index,Letter))),goal_true)),
   add(act(Me,confirm_mutual(next_letter(letter(Index,Letter))),true)),
   del(believe(Me,next_letter(letter(Index,Letter)),mutually_known_true)) ]),
 atts([ name(confirmed_last_letter),
   level(domain) ]) ).
```

```
op(     % goal_request_next_subsequence
 if([ conversants(Me,Other),
  act(Me,confirm_mutual(sequence(Sequence)),goal_true),
  believe(Me,next_subsequence(subsequence(Index,[blank|R])),true) ]),
 not([ act(Me,request(act(Other,assert(subsequence(Index,[blank|R]),Me),
                goal_true),Other),goal_true),
      act(Me,request(act(Other,assert(subsequence(Index,[blank|R]),Me),
                goal_true),Other),true) ]),
 test([]),
 action([]),
 effects([
  add(act(Me,request(act(Other,assert(subsequence(Index,[blank|R]),Me),
                goal_true),Other),goal_true)) ]),
 atts([ level(domain),
  name(goal_request_next_subsequence) ]) ).


op(     % obtained_next_subsequence
 if([ conversants(Me,Other),
  believe(Me,sequence(Sequence),goal_mutually_known_true),
  believe(Me,next_subsequence(subsequence(Index,List)),true),
  act(Me,request(act(Other,assert(subsequence(Index,List),Me),true),
                Other),true),
  act(Me,request(act(Other,assert(next_letter(letter(L_index,Char)),Me),
                true),Other),true) ]),
 not([]),
 test([ length(List,Length),
  L_index >= Length,
  get_next_subsequence(Sequence,
                subsequence(Index,List),
                subsequence(New_index,New_list)) ]),
 action([]),
 effects([ del(believe(Me,next_subsequence(subsequence(Index,List)),true)),
  del(act(Me,request(act(Other,assert(subsequence(Index,List),Me),true),
                Other),true)),
  del(act(Me,request(act(Other,assert(next_letter(letter(L_index,Char)),Me),
                true),Other),true)),
  add(believe(Me,next_subsequence(New_index,New_list),true)) ]),
 atts([ level(domain),
  name(obtained_next_subsequence) ]) ).
```

```
op(     % obtained_last_subsequence
 if([ conversants(Me,Other),
   believe(Me,sequence(Sequence),goal_mutually_known_true),
   believe(Me,next_subsequence(subsequence(Index,List)),true),
   act(Me,request(act(Other,assert(subsequence(Index,List),Me),true),Other),
                  true),
   act(Me,request(act(Other,assert(next_letter(letter(L_index,Char)),Me),
                  true),Other),true) ]),
 not([]),
 test([ length(List,Len1),
   length(Sequence,Len2),
   L_index >= Len1,
   Index + Len1 >= Len2 ]),
 action([]),
 effects([ del(believe(Me,next_subsequence(subsequence(Index,List)),true)),
   del(act(Me,request(act(Other,assert(subsequence(Index,List),Me),true),
          Other),true)),
   del(act(Me,request(act(Other,assert(next_letter(letter(L_index,Char)),Me),
                  true),Other),true)) ]),
 atts([ level(domain),
   name(obtained_next_subsequence) ]) ).


op(     % obtained_next_letter
 if([ conversants(Me,Other),
   act(Me,request(act(Other,assert(subsequence(_,List),Me),goal_true),
                  Other),goal_true),
   act(Other,assert(next_letter(letter(Index,Char)),Me),true),
   believe(Me,next_letter(letter(Index,blank)),true),
   believe(Other,next_letter(letter(Index,Char)),true) ]),
 not([]),
 test([ length(List,Length),
   Index < Length,
   New_index is Index + 1 ]),
 action([]),
 effects([
   del(act(Other,assert(next_letter(letter(Index,Char)),Me),true)),
   del(believe(Me,next_letter(letter(Index,blank)),true)),
   del(believe(Other,next_letter(letter(Index,Char)),true)),
   add(believe(Me,next_letter(letter(New_index,blank)),true)) ]),
 atts([ level(domain),
   name(obtained_next_letter) ]) ).
```

```
op(      % obtained_last_letter
 if([ conversants(Me,Other),
   act(Me,request(act(Other,assert(subsequence(_,List),Me),goal_true),
                    Other),goal_true),
   act(Other,assert(next_letter(letter(Index,Char)),Me),true),
   believe(Me,next_letter(letter(Index,blank)),true),
   believe(Other,next_letter(letter(Index,Char)),true) ]),
 not([]),
 test([ length(List,Length),
        Index >= Length ]),
 action([]),
 effects([
   del(act(Other,assert(next_letter(letter(Index,Char)),Me),true)),
   del(believe(Me,next_letter(letter(Index,blank)),true)),
   del(believe(Other,next_letter(letter(Index,Char)),true)) ]),
 atts([ level(domain),
   name(obtained_last_letter) ]) ).


op(      % goal_assert_next_subsequence_1
 if([    conversants(Me,Other),
         believe(Me,sequence(Sequence),goal_mutually_known_true),
     act(Other,request(act(Me,assert(subsequence(Index,[blank|Rest]),Other),
                    goal_true),Me),true),
         believe(Me,next_subsequence(subsequence(Old_Index,Letters)),true) ]),
 not([ act(Me,assert(believe(Me,New_subseq,true),Other),goal_true),
                    act(Me,confirm_mutual(subsequence(Index,_)),goal_true) ]),
 test([ get_subsequence(Index,sequence(Sequence),New_subseq) ]),
 action([]),
 effects([ del(act(Other,request(act(Me,assert(subsequence(Index,
                    [blank|Rest]),Other),goal_true),Me),true)),
         add(act(Me,assert(believe(Me,New_subseq,true),Other),goal_true)),
         del(believe(Me,next_subsequence(subsequence(Old_Index,Letters)),true)),
         add(believe(Me,next_subsequence(New_subseq), true)) ]),
 atts([ level(domain),
         name(goal_assert_next_subsequence_1) ]) ).
```

```
op(     % goal_assert_next_subsequence_2
 if([    conversants(Me,Other),
         believe(Me,sequence(Sequence),goal_mutually_known_true),
     act(Other,request(act(Me,assert(subsequence(Index,[blank|Rest]),Other),
                 goal_true),Me),true),
         act(Me,confirm_mutual(subsequence(Index,_)),goal_true),
         act(Me,confirm_mutual(next_letter(Letter)),goal_true),
         act(Me,assert(next_letter(Letter),Other),goal_true),
         believe(Me,next_subsequence(subsequence(Old_Index,Letters)),true) ]),
 not([ act(Me,assert(believe(Me,New_subseq,true),Other),goal_true) ]),
 test([ get_subsequence(Index,sequence(Sequence),New_subseq) ]),
 action([]),
 effects([ del(act(Other,request(act(Me,assert(subsequence(Index,
                 [blank|Rest]),Other),goal_true),Me),true)),
         del(act(Me,confirm_mutual(subsequence(Index,_)),goal_true)),
         del(act(Me,confirm_mutual(next_letter(Letter)),goal_true)),
         del(act(Me,assert(next_letter(Letter),Other),goal_true)),
         add(act(Me,assert(believe(Me,New_subseq,true),Other),goal_true)),
         del(believe(Me,next_subsequence(subsequence(Old_Index,Letters)),true)),
         add(believe(Me,next_subsequence(New_subseq), true)) ]),
 atts([ level(domain),
         name(goal_assert_next_subsequence_2) ]) ).


op(     % asserted_next_subsequence
 if([ conversants(Me,Other),
         believe(Me,sequence(Sequence),goal_mutually_known_true),
         believe(Me,next_subsequence(subsequence(Index,List)),true),
         act(Me,assert(believe(Me,subsequence(Index,List),true),Other),true) ]),
 not([]),
 test([ length(List,Len1),
         length(Sequence,Len2),
         Index + Len1 =< Len2,
         get_next_subsequence(Sequence,
                 subsequence(Index,List),
                 subsequence(Next_index,Next_list)) ]),
 action([]),
 effects([ del(believe(Me,next_subsequence(subsequence(Index,List)),true)),
         del(act(Me,assert(believe(Me,subsequence(Index,List),true),Other),
                 true)),
         add(believe(Me,next_subsequence(subsequence(Next_index,Next_list)),
                 true)) ]),
 atts([ level(domain),
         name(asserted_next_subsequence) ]) ).
```

```
op(     % asserted_last_subsequence
 if([ conversants(Me,Other),
        believe(Me,sequence(Sequence),goal_mutually_known_true),
        believe(Me,next_subsequence(subsequence(Index,List)),true),
        act(Me,assert(believe(Me,subsequence(Index,List),true),Other),true) ]),
 not([]),
 test([ length(List,Len1),
        length(Sequence,Len2),
        Index + Len1 > Len2 ]),
 action([]),
 effects([ del(believe(Me,sequence(Sequence),goal_mutually_known_true)),
        add(believe(Me,sequence(Sequence),mutually_known_true)),
        del(believe(Me,next_subsequence(subsequence(Index,List)),true)),
        del(act(Me,assert(believe(Me,subsequence(Index,List),true),Other),
                true)) ]),
 atts([ level(domain),
        name(asserted_last_subsequence) ]) ).

op(     % informed_of_next_subsequence_by_other
 if([ conversants(Me,Other),
        believe(Me,sequence(Sequence),goal_mutually_known_true),
        believe(Me,next_subsequence(subsequence(Index,[blank|Rest])),true),
        act(Other,assert(believe(Other,subsequence(Index,List),true),Me),
                true) ]),
 not([]),
 test([ length([blank|Rest],Len1),
        length(Sequence,Len2),
        Index + Len1 =< Len2,
        get_next_subsequence(Sequence,
                subsequence(Index,[blank|Rest]),
                subsequence(Next_index,Next_list)) ]),
 action([]),
 effects([ del(believe(Me,next_subsequence(subsequence(Index,[blank|Rest])),
                true)),
        del(act(Other,assert(believe(Other,subsequence(Index,List),true),Me),
                true)),
        add(believe(Me,next_subsequence(subsequence(Next_index,Next_list)),
                true)) ]),
 atts([ level(domain),
        name(informed_of_next_subsequence_by_other) ]) ).
```

```
op(     % informed_of_last_subsequence_by_other
 if([ conversants(Me,Other),
        believe(Me,sequence(Sequence),goal_mutually_known_true),
        believe(Me,next_subsequence(subsequence(Index,[blank|Rest])),true),
        act(Other,assert(believe(Other,subsequence(Index,List),true),Me),
                true) ]),
 not([]),
 test([ length([blank|Rest],Len1),
        length(Sequence,Len2),
        Index + Len1 > Len2 ]),
 action([]),
 effects([ del(believe(Me,sequence(Sequence),goal_mutually_known_true)),
        add(believe(Me,sequence(Sequence),mutually_known_true)),
        del(believe(Me,next_subsequence(subsequence(Index,[blank|rest])),
                true)),
        del(act(Other,assert(believe(Other,subsequence(Index,List),true),Me),
                true)) ]),
 atts([ level(domain),
        name(informed_of_last_subsequence_by_other) ]) ).
```

APPENDIX J

RULE-BASED SYSTEM: DOMAIN-SPECIFIC CLAUSES

```
% Meta-locutionary rule-based system -- domain specific clauses
% DGN 10-18-88


%%%%%
%%%%% Program: Domain-specific clauses
%%%%%


get_subsequence(Index,sequence(Sequence),subsequence(Index,New_list)) :- !,
 same_type_sublist(Sequence,New_list,Index).

get_next_subsequence(Sequence,
          subsequence(Old_index,Last_list),
       subsequence(New_index,Next_list)) :- !,
          length(Last_list,Length),
 New_index is Old_index + Length,
 same_type_sublist(Sequence,Next_list,New_index).

same_type_sublist(List,Sublist,Index) :- !,
 sublist(List,Temp_sublist,Index),
 complete_same_type_sublist(Temp_sublist,Sublist).

sublist(L,L,1) :- !.
sublist([H|T],S,Index) :- !,
 J is Index - 1,
  sublist(T,S,J).

complete_same_type_sublist([blank|T1],[blank|T2]) :- !,
 complete_blank_sublist(T1,T2,5).
complete_same_type_sublist([Char|T1],[Char|T2]) :- !,
 complete_char_sublist(T1,T2,5).

complete_blank_sublist(_,[],1) :- !.
complete_blank_sublist([blank|T1],[blank|T2],Index) :- !,
 J is Index - 1,
 complete_blank_sublist(T1,T2,J).
complete_blank_sublist(_,[],_).

complete_char_sublist(_,[],1) :- !.
complete_char_sublist([],[],_) :- !.
complete_char_sublist([blank|T1],[],_) :- !.
complete_char_sublist([Char|T1],[Char|T2],Index) :- !,
 J is Index - 1,
 complete_char_sublist(T1,T2,J).

get_next_letter(Index,List,letter(New_index,New_char)) :- !,
 New_index is Index + 1,
 nth(New_index,List,New_char).
```

APPENDIX K

RULE-BASED SYSTEM:
INITIAL STATE

```
% Meta-locutionary rule-based system --  domain initial state
% DGN 10-18-88


%%%%%
%%%%%  Knowledge: base facts
%%%%%


cycle(0).                                   % Initial value

person(adam).                                      % The two conversants
person(barney).

conversants(adam,barney).


% Initial state
%

:- recorda(adam,
  state(conversants(adam,barney),0,-1),
  _).

:- add(adam,believe(adam,turn(barney),mutually_known_true)).

:- add(adam,(act(adam,confirm_mutual(sequence([blank,i,s,u,t,w,r,q,g,l,d,
          blank,blank,blank,blank,g,o])),goal_true))).

:-  add(adam,believe(adam,sequence([blank,i,s,u,t,w,r,q,g,l,d,blank,
          blank,blank,blank,g,o]),goal_mutually_known_true)).

:- add(adam,believe(adam,next_subsequence(subsequence(1,[blank])),true)).

:- add(adam,act(barney,give_turn(adam),true)).

:- recorda(barney,
  state(conversants(barney,adam),
       0, -1),
  _).

:- add(barney,believe(barney,turn(adam),true)).

:- add(barney,act(barney,confirm_mutual(sequence([o,blank,s,u,t,w,r,q,
          g,blank,blank,f,w,w,d,g,o])),goal_true)).

:-  add(barney,believe(barney,sequence([o,blank,s,u,t,w,r,q,g,
        blank,blank,f,w,w,d,g,o]),goal_mutually_known_true)).

:- add(barney,believe(barney,next_subsequence(subsequence(1,[o])),true)).
```

BIBLIOGRAPHY

Allen, A. E., & Guy, R. F. (1974). *Conversation analysis.* The Hague: Mouton.

Allen, E. (1983). *YAPS: yet another production system* (Tech. Rep. No. 1146). College Park, MD: University of Maryland, Computer Science Department.

Allen, J., & Perrault, C. R. (1980). Analyzing intentions in utterances. *Artificial Intelligence, 15*, 143-178.

Appelt, D. E. (1981). *Planning natural language utterances to satisfy multiple goals* (Tech. Rep.). Palo Alto, CA: SRI.

Appelt, D. E. (1985). Planning English referring expressions. *Artificial Intelligence, 26*, 1-33.

Argyle, M., & Cook, M. (1976). *Gaze and mutual gaze.* Cambridge: Cambridge University Press.

Argyle, M., Ingham, R., Alkema, F., & McCallin, M. (1981). The different functions of gaze. In A. Kendon (Ed.), *Nonverbal communication, interaction, and gesture* (pp. 283-295). The Hague: Mouton. (Reprinted from *Semiotica*, 1973, *7*, 19-32.)

Austin, J. L. (1962). *How to do things with words.* New York: Oxford University Press.

Bach, K., & Harnish, R. M. (1979). *Linguistic communication and speech acts.* Cambridge, MA: The MIT Press.

Beattie, G. W. (1981). Sequential temporal patterns of speech and gaze in dialogue. In A. Kendon (Ed.), *Nonverbal communication, interaction, and gesture* (pp. 297-320). The Hague: Mouton. (Reprinted from *Semiotica*, 1978, *23*, 27-52.)

Birdwhistell, R. L. (1970). *Kinesics and context.* Philadelphia: University of Pennsylvania Press.

Birnbaum, L. A. (1986). Integrated processing in planning and understanding (Doctoral Dissertation, Yale University, 1981). *Dissertation Abstracts International, 48*, 3023B.

Bowers, J., & Churcher, J. (1988). Local and global structuring of computer mediated communications: developing linguistic perspectives on CSCW in COSMOS. *Proceedings of the Conference on Computer-Supported Cooperative Work* (pp.125-139). New York: The Association for Computing Machinery, Inc.

Chafe, W. (1986). Cognitive constraints on information flow. In R. Tomlin (Ed.), *Coherence and grounding in discourse* (pp. 21-51). Amsterdam: J. Benjamins.

Chapanis, A., Ochsman, R. B., Parrish, R.N., & Weeks, G. D. (1972). Studies in interactive communication: I. The effects of four communication modes on the behavior of teams during coöperative problem solving. *Human Factors, 14*, 101-125.

Chapanis, A., Ochsman, R. B., Parrish, R.N., & Weeks, G. D. (1977). Studies in interactive communication: II. The effects of four communication modes on the linguistic performance of teams during coöperative problem solving. *Human Factors, 19*, 487-509.

Charniak, E. (1984). cognitive science is methodologically fine. In W. Kintsch, J. R. Miller, & P. G. Polson (Eds.), *Methods and tactics in cognitive science* (pp. 263-274). Hillsdale, NJ: Lawrence Erlbaum Associates.

Clark, H. H., & Marshall, C. R. (1981). Definite reference and mutual knowledge. In A. K. Joshi, B. L. Webber, & I. A. Sag (Eds.), *Elements of discourse understanding* (pp. 10-63). New York: Cambridge University Press.

Clark, H. H., & Schaefer, E. F. (in press). Contributing to discourse. *Cognitive Science*.

Clark, H. H., & Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition, 22*, 1-39.

Cohen, P. R. (1981). The need for referent identification as a planned action. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 31-35). Los Altos, CA: William Kaufman, Inc.

Cohen, P. R. (1984). The pragmatics of referring and the modality of communication. *Computational Linguistics, 10* (2), 97-146.

Cohen, P. R. & Levesque, H. J. (1986). Speech acts and rationality. *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. Morristown, NJ: Association for Computational Linguistics.

Cohen, P. R., & Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science, 3* (3), 177-212.

Deitel, H. M. (1984). *An introduction to operating systems*. Reading, MA: Addison-Wesley.

De Long, A. J. (1983). Kinesic signals at utterance boundaries in preschool children. In A. Kendon (Ed.), *Nonverbal communication, interaction, and gesture* (pp. 251-281). The Hague: Mouton. (Reprinted from *Semiotica*, 1974, *11*, 43-74.)

Douglas, S. A. (in press). Tutoring as interaction. In P. Goodyear (Ed.), *Teaching Knowledge and Intelligent Tutoring*. Norwood, NJ: Ablex Publishing Corporation.

Douglas, S. A., Novick, D. G., & Olivier, S. P. (1987). *Discourse management in a tutoring environment.* Unpublished manuscript, University of Oregon, Department of Computer and Information Science, Eugene, OR.

Duncan, S., Jr. (1980). Describing face-to-face interaction. In W. Von Raffler-Engel (Ed.), *Aspects of nonverbal communication* (pp. 67-80). Lisse, Germany: Swets and Zeitlinger.

Ekman, P. (1980). Three classes of nonverbal behavior. In W. Von Raffler-Engel (Ed.), *Aspects of nonverbal communication* (pp. 89-102). Lisse, Germany: Swets and Zeitlinger.

Ekman, P., & Friesen, W. V. (1981). The repetoire of nonverbal behavior. In A. Kendon (Ed.), *Nonverbal communication, interaction, and gesture* (pp. 57-105). The Hague: Mouton. (Reprinted from *Semiotica*, 1969, *1*, 49-97.)

Fickas, S. F., & Novick, D. G. (1985). Control knowledge in expert systems: relaxing restrictive assumptions. *Proceedings of the Fifth International Conference on Expert Systems and Their Applications* (pp. 981-994). Avignon, France: Agence de l'Informatique.

Forgy, C. (1981). *OPS5 user's manual.* Pittsburgh: Carnegie-Mellon University, Department of Computer Science.

Frederking, R. E. (1988). *Integrated natural language dialogue*. Norwell, Mass.: Kluwer Academic Publishers.

Givón, T. (1984). Deductive vs. pragmatic processing in natural language. In W. Kintsch, J. R. Miller, & P. G. Polson (Eds.), *Methods and tactics in cognitive science* (pp. 137-190). Hillsdale, NJ: Lawrence Erlbaum Associates.

Goodman, B. A. (1986). Reference identification and reference identification failures, *Computational Linguistics, 12* , 273-305.

Grice, H. P. (1975). Logic and conversation. In P. Cole, & J. L. Morgan (Eds.), *Studies in syntax*, vol. 3 (pp. 225-242). New York: Seminar Press.

Grosz, B. J. (1977). The representation and use of focus in a system for understanding dialogs. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 67-76). Los Altos, CA: William Kaufman, Inc.

Grosz, B. J. (1981). Focusing and description in natural language dialogues. In A. K. Joshi, B. L. Webber, & I. A. Sag (Eds.), *Elements of discourse understanding* (pp. 84-105). New York: Cambridge University Press.

Grosz, B. J. (1982). Discourse analysis. In R. Kittredge, & J. Lehrberger (Eds.), *Sublanguage: studies of language in restricted semantic domains* (pp. 138-174). Berlin: Walter de Gruyter.

Grosz, B. J., & Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, *12*, 175-204.

Hayes, P., & Reddy, D. R. (1983). Steps toward graceful interaction in spoken and written man-machine communication. *Interational Journal of Man-Machine Studies*, *19*, 231-284.

Hobbs, J. R. (1985). *On the coherence and structure of discourse* (Report No. CSLI-85-37). Stanford, CA: Center for the Study of Language and Information.

Hobbs, J. R., & Evans, D. A. (1980). Conversation as planned behavior. *Cognitive Science*, *4*(4), 349-377.

Hoffer, B., & Santos, R. G. (1983). Cultural clashes in kinesics. In W. Von Raffler-Engel (Ed.), *Aspects of nonverbal communication* (pp. 335-338). Lisse, Germany: Swets and Zeitlinger.

Hopson-Hasham, B. (1983). The adjustment of the speaker to the hearer. In W. Von Raffler-Engel (Ed.), *Aspects of nonverbal communication* (pp. 281-286). Lisse, Germany: Swets and Zeitlinger.

Hovy, E. H. (1985). Integrating text planning and production in generation. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, (pp.848-851). Los Altos, CA: Morgan Kaufman Publishers, Inc.

Jernudd, B. H., & Thuan, E. (1983). Control of language through correction in speaking, *International Journal of the Sociology of Language*, *44*, 71-97.

Johnson, P. N., & Robertson, S. P. (1981). *MAGPIE: a goal-based model of conversation* (Research Report No. 206). New Haven: Yale University, Department of Computer Science.

Levinson, S. C. (1981). The essential inadequacies of speech act models of dialogue. In H. Parret, M. Sbisa, & J. Verschuren (Eds.), *Possibilities and limitations of pragmatics: Proceedings of the Conference on Pragmatics at Urbino*, July 8-14, 1979 (pp. 473-492). Amsterdam: Benjamins.

Kendon, A. (1981). Introduction: Current issues in the study of "nonverbal communication." In A. Kendon (Ed.), *Nonverbal communication, interaction, and gesture* (pp. 1-53). The Hague: Mouton.

McDermott, J., & Forgy, C. (1978). Production system conflict resolution strategies. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems* (pp. 177-199). New York: Academic Press.

McKeown, K. R. (1985). *Text generation.* New York: Cambridge University Press.

Miller, J. R., Polson, P.G., & Kintsch, W. (1984). Problems of methodology in cognitive science. In W. Kintsch, J. R. Miller, & P. G. Polson (Eds.), *Methods and tactics in cognitive science* (pp. 1-18). Hillsdale, NJ: Lawrence Erlbaum Associates.

Novick, D. G. (1986). [Transcription of tutor-student protocol for second-language tutoring session]. Unpublished raw data. University of Oregon, Department of Computer and Information Science, Eugene, OR.

Perrault, C. R., & Allen, J. F. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, *6*(3), 167-182.

Power, R. (1979). The organization of purposeful dialogues. *Linguistics*, *17*, 107-152.

Reichman, R. (1985). *Getting computers to talk like you and me.* Cambridge, MA: MIT Press.

Sacks, H., Schegloff, E., & Jefferson, G. (1974). A simplest systematics for the organization of turn-taking in conversation. *Language*, *50*, 696-735.

Sanford, D. L., & Roach, J. W. (1987). Representing and using metacommunication to control speakers' relationships in natural language dialogue. *International Journal of Man-Machine Sudies*, *26*, 301-319.

Scheflen, A. E. (1980). Systems in human communication. In W. Von Raffler-Engel (Ed.), *Aspects of nonverbal communication* (pp. 7-28). Lisse, Germany: Swets and Zeitlinger.

Schegloff, E. A., Jefferson, G., & Sacks, H. (1977). The preference for self-correction in the organization of repair in Conversation. *Language*, *33*, 361-382.

Searle, J. R. (1969). *Speech acts*. Cambridge: Cambridge University Press.

Simon, H.A. (1972). Complexity and the representation of patterned sequences of symbols. *Psychological Review*, 79, 369-382.

Simon, H. A. (1974). How big is a chunk? *Science*, *183*, 482-488.

Stucky, S. U. (1988). The situated process of situated language. Unpublished manuscript, Xerox PARC, Palo Alto, CA.

Suchman, L. A. (1987). *Plans and situated actions*. Cambridge: Cambridge University Press.

Suppes, P. (1984). Some general remarks on the cognitive sciences. In W. Kintsch, J. R. Miller, & P. G. Polson (Eds.),*Methods and tactics in cognitive science* (pp. 297-304). Hillsdale, NJ: Lawrence Erlbaum Associates.

Swedish Institute of Computer Science (1988). *SICStus Prolog user's manual*. Kista, Sweden: Swedish Institute of Computer Science.

Swinney, D. (1984). Theoretical and methodological issues in cognitive science: a psycholinguitic perspective. In W. Kintsch, J. R. Miller, & P. G. Polson (Eds.), *Methods and tactics in cognitive science* (pp. 217-233). Hillsdale, NJ: Lawrence Erlbaum Associates.

VanLehn, K., Brown, J. S., & Greeno, J. (1984). Competitive argumentation in computational theories of cognition. In W. Kintsch, J. R. Miller, & P. G. Polson (Eds.), *Methods and tactics in cognitive science* (pp. 235-262). Hillsdale, NJ: Lawrence Erlbaum Associates.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition.* Norwood, NJ: Ablex Publishing Corporation.