Mixed Initiative Dialogue and Intelligence via Active Logic

Carl Andersen David Traum K. Purang Darsana Purushothaman
Don Perlis
University of Maryland
A.V. Williams Building
College Park, MD 20742 USA
phone: +1 (301) 405-1139
fax: +1 (301) 405-6707
{cfa,traum,kpurang,darsana,perlis}@cs.umd.edu

Abstract

This paper describes features of the active logic formalism as a tool for implementing mixed initiative intelligent systems. A framework is provided for assessing systems as "mixed-initiative", which is used to explore the relevant features both in the active logic formalism and associated implementations based upon and using that formalism. Active logics were developed as a means of combining the best of two worlds – inference and reactivity – without giving up much of either. This requires a special evolving-during-inference model of time. Active logics are able to react to incoming information (including dialogue utterances by a collaborative partner) while reasoning is ongoing, blending new inputs into its inferences without having to start up a new theorem-proving effort. An implementation of active logic is described, which also includes special features for reasoning about and performing actions. This implementation is also used as the backbone of a dialogue system.

1 Introduction

The term *mixed-initiative* can mean many different things to different researchers/sub-groups. A minimal requirement is at least that "initiatives" (whatever they are) come from multiple sources. A more restrictive definition would also require that the pattern of initiatives or right to produce initiative (or the commodity of "the initiative", itself, whatever that implied) is not fixed in advance, but develops through an interaction. Using this more restrictive definition, an ability to engage in mixed initiative interaction puts certain constraints on an intelligent system (or theoretical specification of such a system). Among the required abilities are:

Reactivity A mixed-initiative system must be able to react to new inputs in the course of its action or cogitation. While this is a desirable ability for flexible systems in general, a system must at least have the ability to be able to recognize and react appropriately to initiatives coming from a partner (be that a human user or other computer system/agent). A mixed-initiative system should not just "turn-off" or buffer it's input for long periods of time, making it unavailable for communicative interaction, while working on a particular task.

(limited) autonomy In order to engage in mixed-initiative behavior, a system will have to actually *take* the initiative from time to time, as well as being sensitive to the initiatives of another. A "Slave" system that merely fulfills requests of a user or other system, no matter how reactive or helpful it might be, could not be described as a *mixed-initiative* system. The ability to meaningfully take the initiative implies a certain kind of orientation toward goals (however they might be internally represented) which might over-ride faithful reactions to the initiatives of others. This autonomy should not be absolute, however, since the system will still need to be somewhat reactive and sensitive to the initiatives of others.

communicative competence In order to act in a way that might be described as mixed-initiative, a system must be able to understand (in some sense) the initiatives of others, and to provide initiatives in such a way that they can be understood. If perfect understandability can not be guaranteed, then there should also be some mechanism for *grounding* [Clark and Schaefer, 1989, Traum, 1994] the communications, or recovering from mis-understanding [McRoy and Hirst, 1995].

collaborative competence mixed initiative behavior also requires some ability to collaborate with another. At a bare minimum, there must be collaboration over management of joint resources, such as the ability to take initiatives. This requires some notion of commitments [Cohen and Levesque, 1990] and perhaps joint intentions [Cohen and Levesque, 1991], SharedPlans [Grosz and Sidner, 1990] and/or obligations [Traum and Allen, 1994].

negotiative ability as well as the ability to collaborate, mixed initiative systems must have a capacity for both forming new collaborative partnerships or tasks, as well as arbitrating between conflicting goals. Key, at least is the ability to negotiate local initiative, but further ability to actually negotiate authority over decisions will be useful as well.

Very few computer systems (or even AI systems or natural language dialogue systems) would qualify as "mixed initiative systems" according to the above principles. Most leave out one or more of the above abilities. For example classical planning systems are generally not "reactive", in the sense that, given a goal and initial state, they just produce a plan with no more interaction from a user or environment (though most plan execution systems would at least be reactive to the environment in trying to apply plans). Other planning frameworks, e.g., [Ferguson, 1995, Lochbaum, 1994] could be described as having reactivity and some communicative or collaborative competence, since they can elegantly represent interleaved planning and plan-recognition, but might lack the kind of autonomy or negotiative ability to decide when to put forth an initiative rather than accept one. Likewise, natural language dialogue systems will have some amount of communicative competence, but may also lack autonomy or negotiative ability. Agents within a multi-agent system will generally have a degree of autonomy and reactivity, but may lack the other abilities.

In this paper, we describe our current research in the direction of mixed-initiative intelligence. We start, in the next section with a description of *Active Logic*, an approach to logic and inference that is more amenable to mixed-initiative intelligence than most classical logics. In Section 3, we consider features of Active logic that facilitate mixed-initiative intelligence. In section 4, we describe an implementation of Active Logic which allows for the programming of a mixed-intelligence agent. We continue in section 5, with a description of how the core of this agent is used to construct a dialogue manager for a natural language dialogue system. Finally, we conclude in Section 6 with a discussion of additional features beyond the core abilities provided by the implementation, to achieve mixed initiative interaction and a description of relevant future project goals.

2 Active Logic

Active logics are a family of formalisms undergoing development, theoretical analysis, implementation and testing at the University of Maryland. One of the original motivations for active logics was that of designing formalisms for reasoning about an approaching deadline; for this use it is crucial that the reasoning take into account the ongoing passage of time as that reasoning proceeds.

To achieve this, active logics employ a notion of "now" that is constantly updated by an inference rule, whose "firing" is strictly regulated. Theorems are marked as to the time of being proven, ie, the current value of "now", and this time-marker is itself something that further inferences can depend on, such as inferring that a given deadline is now too close to meet by means of a particular plan under refinement, if its enactment is estimated to take longer than the (ever shrinking) time remaining before the deadline.

Such formalisms are distinct from traditional temporal logics, in that the latter characterize truth about past, present, and future as if from a timeless (or unchanging) present; that is, the inferences do not correspond to an increase in the value of "now". This is appropriate as long as the temporal reasoning is by one agent about another agent far removed in time, or if the latter agent's activity is independent of the former. But when an agent is reasoning about its own ongoing activity, or about another agent whose activity is highly interdependent, traditional "time-frozen" reasoning is at a disadvantage, and "time-tracking" active logics can bring new power and flexibility to bear.

Active logic formalisms are essentially non-monotonic; however they do not proceed from one non-monotonic theory (set of axioms) at step i to a new non-monotonic theory (updated set of axioms) at step i + 1. Instead, there is the notion of a unique discretely evolving theory. The incorporation of new information is adjudicated as an inference process, including the decision as to which "axioms" and "inferences" are to be retained and which are to be rejected. The temporal history of the reasoning process together with a quotation mechanism, enables the formalism to reason about its own reasoning in time.

The belief state transitions are effected by a set of rules that map the history up to time t on to a new state at t+1. For instance, by default, the inheritance rule will cause all the beliefs at time t to be inherited to time t+1. However, in the presence of direct contradictions at time t, (ie., both p and not (p) are beliefs at time t), neither of the beliefs is inherited to time t+1. This in effect removes the contradiction from t+1 belief state and prevents it from directly affecting future states. Such a contradiction will also trigger the belief contra(p,not(p)) at t+1 to indicate that a contradiction occurred. The logic can then reason about the contradictants and take appropriate actions including reinstating the belief judged correct (if any). In this sense, active logic formalisms mirror common sense agents in handling contradictions.

We have explored a number of aspects of active logics in some detail, from the basic concepts [Elgot-Drapkin and Perlis, 1990], to deadline reasoning [Kraus *et al.*, 1990, Nirkhe, 1994, Nirkhe *et al.*, 1997], to reasoning about others' ongoing reasoning ([Elgot-Drapkin, 1988, Elgot-Drapkin, 1991]), and have begun to explore uses of active logics in dialogue, especially in pragmatic inferences that support dialog ([Gurney *et al.*, 1995, Perlis *et al.*, 1996]).

3 Active logic as resource for reasoning about mixed initiative interaction

The two previous sections outlined our general view of mixed initiative, and the general character of active logics. Now we discuss ways in which active logic lends itself to mixed initiative.

Reactivity and autonomy: Active logics are inherently reactive and autonomous. That is, they accept, and process in real time, incoming signals along with their intrinsic internal inferencing. Thus an active logic is both responsive to other agents that may send messages to it, and also has its own agenda and can, for instance, decide to ignore particular messages or agents. Active logics can also decide whether to interrupt another agent, or to begin a new topic; that is, the logic can take initiative: it is always in control of how it reacts to events.

This control takes two intertwined forms: deciding what to do, and introspecting on what it has done. These two forms correspond loosely to reactivity and autonomy as well. In many cases, a decision is made simply on the basis of existing information about the external world: X and Y are the case, so do Z; this corresponds to the behavior of a largely traditional theorem prover, except that updated (current) times can play a major role in such inferences, as noted above in the introduction.

However, internal state information also can play a role in inferences, such as the state of the logic's own goals (are they achieved yet? are they being worked on? will they be served by a particular action?) A simple version of this is negative introspection, often encoded as the following inference rule:

```
i:        ...
        ----------
i+1:    -Know(i,B)
```

which mandates the conclusion at time i+1 that statement B was not known to the logic (not among its beliefs) at time i (where B does not appear among those beliefs, shown as ...). For instance, if Doing(A) is not a belief at time i, then that fact may be inferred at time i+1, and the logic may then decide to initiate action A at time i+2, say; whereas if it was already executing A, it would have access (positive introspection) to that fact and might refrain from re-executing A.

Communication, Collaboration and negotiation: A key aspect of understanding communication involves not just recognizing *what* was said, but *why* it was said, and what it reveals about the internal state of the speaker and what the effects on the hearer are supposed to be (i.e., the *expressive* and *evocative* communicative functions of [Allwood *et al.*, 1989]). A representation of dialogue context and an ability for introspection or simulation of agents can be very useful here. Moreover, an historical record of who said what, and when and in what context, can be essential in deciding who should take initiative next. Active logics automatically provide an historical record of such time-coupled contexts; essentially this is simply a record of everything that has passed through the evolving belief base, including its own inferences as well as observations (incoming data). Future inferences can then avail themselves of this record, deciding for instance that it is Smith's turn to talk since a question has just been put to Smith; or that since Smith already answered that question ten minutes ago, the logic can supply the answer directly, or can inform the questioner that it has already been answered.

One example of the above that we have been investigating is that of unwarranted repetitions. If the same utterance is made several times, there is probably a special significance to the repetition; it could be for emphasis, or to signal that it has not been properly responded to. In any case, the system should recognize the fact of repetition, which the active-logic history facilitates. Other dynamic logics (e.g., [Harel, 1979, Cohen and Levesque, 1990, Sadek, 1992]) allow convenient representation of an event that has just happened, but are less convenient at representing repetitions or sequences of similar actions.

4 Alma and Carne: An Active Logic Agent

As we have described in the previous sections, Active Logic has several features which are more amenable to the description of mixed-initiative intelligence than more standard logics. These features also make it well suited to serve as the backbone of a logical-reasoning agent. Not only is active logic better suited to implementation of a reasoning agent, which will, of necessity be resource-bounded, it also has advantages over standard logic-programming paradigms, such as prolog. As with the interruptible prolog, IPSIM, described by [Smith and Hipp, 1994], an agent based on active logic will have the ability to adapt the reasoning to initiative demands of the interaction, rather than just doing full proofs without interruption. Another difficulty with standard prolog is the lack of distinction between procedural knowledge and action execution on the one hand, and logical inference on the other.

We have designed and implemented an "agent-programming" system, using active logic as the inference model. This system makes a clear distinction between factual knowledge, on the one hand, and performance of non-logical actions, on the other. In this way, standard procedural programming techniques are available without muddying the semantics of inference, as is common in traditional logic-programming languages. This can be particularly problematic when partial proofs have side-effects. An interface between the two components allows reasoning about action, as well as "decisions" to invoke actions. In this section we describe the basic features of this programming system. In the next, we describe how this system is used as the core of a dialogue manager.

Our Agent programming system consists of two distinct but connected sub-components, Alma, for Active Logic MAchine – also "spirit" in Spanish) a logical reasoner, the other (Carne, the "flesh" to Alma's "spirit") an action executor, which is able to run user-defined programs (including standard prolog programs). Built into Alma is the capacity to initiate function calls to Carne which may return information to Alma and/or initiate some external action by sending messages to other modules or agents (using KQML). Carne also serves as a incoming message server for Alma: all messages to the system are routed to Carne, which automatically converts the message KQML representation into a set of equivalent active logic assertions which it passes to Alma. With these capabilities, Carne serves in the capacity of Alma's eyes, voice and hands.

Alma and Carne are implemented as separate processes with Alma the reasoner and Carne the action execution module. This gives us a clear separation between the procedural and the declarative parts of the model of the agent while allowing declarative knowledge about the procedures to be explicitly stated and accessible for time-dependent reasoning.

4.1 Alma

Alma implements active logic and is the repository for declarative knowledge in the agent. All inferences and all decisions to act are done in Alma, controlled by domain axioms and active logic rules of inference described above. The Alma implementation has a few features that enhance the efficiency of the logical inference engine:

- Alma applies its inference rules only to the new formulas derived at each step and includes an operator that can be used in axioms to explicitly remove formulas from the knowledge base (treating everything else as implicitly inherited from one step to the next).
- The formulas in Alma are of three types according to how they are to be used. This constitutes another way one can control the inferences made and the size of the resulting knowledge base.
  - Formulas to be used in forward chaining proofs only.
  - Formulas to be used in backward chaining proofs only.
  - Formulas that can be used for either forward or backward chaining.
- The programmer can set general policies to determine which inferences, out of all those possible at every step, to actually make. This enables one to restrict inferences to "interesting" parts of the knowledge base, though doing so productively requires good domain knowledge .

Alma also has the capability to interact with Carne, as mentioned above and described below, following a description of Carne.

4.2 Carne

Carne contains the procedural knowledge of the reasoner. It allows the programmer to specify programs (in prolog) that fall in these main categories:

- Programs triggered by Alma to effect a change in the environment.
- Programs that are responsive to events in the environment (e.g., receipt of a KQML message from another agent or system module) and automatically update Alma's knowledge base.
- Programs that do (non-logical) computations on behalf of Alma.

This gives Alma the ability to effectively interact with the world and to offload resource intensive computations to a separate process, which is asynchronous with respect to the time progression of Alma.

4.3 The Alma/Carne interface

A simple interface is used to link Alma and Carne.

On the Alma side, there are special purpose inference rules and predicates for invoking and monitoring programs in Carne. These predicates can be used in axioms to initiate programs in Carne, and to reason about the status of the programs.

The predicates are:

$call(\phi, Id)$ If a formula of the form $call(\phi, Id)$ is derived in Alma, an inference rule comes into play that sends a message to the Carne process for it to execute program ϕ (which, of course, has to be known to Carne). The rule also results in the assertion $doing(\phi, Id)$ in Alma's knowledge base. The Id is a unique identifier used to distinguish between multiple invocations of the same program with the same arguments.

$doing(\phi, Id)$ indicates Carne is in the process of executing the action id, using program named ϕ.

$done(\phi, Id)$ Once the program has completed successfully in Carne, a message is sent to Alma that results in the assertion of $done(\phi, Id)$ in Alma's knowledge base and the deletion of $doing(\phi, Id)$ (although the fact that *doing* was true at a particular time remains in the Alma history).

$error(\phi, Id)$ In case the program fails to successfully complete execution in Carne, $error(\phi, Id)$ is added to and $doing(\phi, Id)$ is deleted from the Alma database.

These predicates allow Alma to track the status of the programs executing in Carne and enables decisions to be made about actions as described in the previous section. This general facility for reasoning about action extends beyond just the Alma/Carne interface: the same *doing*, *done* and *error* predicates allow Alma to monitor complex actions, actions established by effects rather than program runs, and actions by other agents as well as multi-agent actions, using the same basic framework. These other actions would not be directly connected to Carne programs in the same way (invoked by *call* predicates), but could be used in the same way to trigger further inference.

On the Carne side, the prolog predicates *af* (add formula) *df* (delete formula) are provided to the Carne programs, allowing them to update the Alma knowledge base. This facility is independent of the above status predicates and is used to assert the results of computations and for input from the environment to be included in Alma in the appropriate form.

This Alma/Carne agent architecture naturally leads to mixed initiative abilities. Inferring a *call* predicate leads to, in some sense, the agent taking the initiative by invoking a Carne program to perform an action. However, simply noticing that an action has been performed (whether by Carne or otherwise, following an initiative of another) can lead to the same kinds of further inferences. Likewise, Carne's ability to observe and directly manipulate Alma's database (e.g., on the basis of received messages) allows inferences to proceed on the basis of self or other produced information.

5 A Dialogue Manager Using Alma and Carne

The Alma-Carne Dialog Manager (ACDM) is the main reasoning module for our dialogue system, building on the basic functions of Alma and Carne, described in the previous section. Our current ACDM is embedded as a module within the TRAINS-96 system [Allen *et al.*, 1996]. This system is designed to engage a human user in natural language communication regarding a train scheduling task. In this section we briefly describe the dialogue system in which this is embedded and the dialogue manager, itself.

5.1 Maryland version of TRAINS-96

The TRAINS-96 system consists of a set of heterogeneous modules communicating through a central hub using messages in KQML[Group, 1993]. This architecture is thus well suited for swapping in different components to do the same or a similar job and assessing the results. As well as the architecture itself, we have been using the parser, domain problem solver, and display modules from the original system, replacing the discourse manager component with our own dialogue manager and multi-modal generator. The functions of the modules in the Maryland version of the system are summarized below:

Parser: produces interpretation of sentence input

Problem Solver: answers queries for problem state, also answers planning requests from dialogue manager

Display Manager: shows objects on screen

Dialogue manager: utterance interpretation, dialogue reasoning, and decision to produce initiatives and responses.

Output Manager: multimodal presentations of system output, including calls to display manager, printed text, and speech.

5.2 The Alma-Carne Dialog Manager

ACDM logically represents several different kinds of information about the dialog it engages in. First, high-level dialog strategies represent user and system goals and other modalities in a description of how dialogs customarily proceed. For example, the system contains a number of rules about reacting to requests from the user. When presented with a user request to "Send the Boston train to New York", the system deduces that the user has a request. Another rule mandates that the system must attempt to fulfill user requests when possible, and deduces a system goal to fulfill this one. Currently, these rules are able to handle only simple request/answer pairings, but in the future, we will expand such rules to handle ongoing dialogs. Such dialogs will require modeling and tracking a complex dialog state composed of multiple simultaneous system/user goals and obligations, as well as sequences of speech acts and their interrelationships. The system will, on a continuing basis, choose which goals to address (or abandon), as well as reason whether to seize or relinquish initiative.

Instrumental to the processing of such strategies are interpretation rules representing system knowledge about particular speech act types; these rules allow the system to both identify encountered user speech acts and formulate system speech act responses. Most of this information classifies various speech acts using details of their logical form. In the above example, initial information from the parser identifies the user utterance as a request, but the ACDM uses further inference to classify details of the request, such as that it requests making a change (moving a train) in the domain, as opposed to simply asking for the answer to a question.

Finally, other facts logically model the complex actions the system may undertake in order to fulfill its higher-level dialog goals. Complex actions consist of a series of subsidiary actions and include required checkpoint observations that ensure the intended result is being achieved. Alma executes these actions in a logical framework, tracking execution status by asserting predicates like $doing(Act1)$ and $done(Act1)$, in a manner parallel to the Alma/Carne interface described in the previous section. All primitive actions which actually affect the world in some manner are calls to Carne procedures, e.g. call(sendreply("No", user)).

To illustrate action reasoning, we continue the example introduced above. ACDM deduces that to fulfill the user's request it must execute a complex action composed of a string of sub-actions. First, ALMA must determine exactly which train denoted by "Boston train" to send to New York. This entails sending a query message to the domain reasoner, which replies (again via Carne) with information disambiguating the train meant by the user, in this case Northstar Express. The action of actually sending the train entails more requests of the domain reasoner: to plan a route for the Northstar Express to New York, and to actually send that train to New York on that route. Once these sub-actions are completed, Alma fulfills its top-level goal by sending a message to the output manager instructing it to inform the user that the Northstar Express has successfully reached New York.

ACDM's declarative representation and its modeling of complex actions make sophisticated strategies for taking and releasing initiative natural to model. ACDM's strength is that it can represent multiple ongoing streams of reasoning, evaluation, and action which automatically interact only in the appropriate circumstances. For example, suppose the user and the system share the goal of finding a path from location A to location B. The system may execute a collaborative action, which will allow different possible ways of executing. The system might take the initiative and make proposals to the user, or instead might surrender initiative to the user. Logically, the choice of which alternative to choose may be linked to both longer and shorter-term goals and obligations of the system. In the default case, the system might simply presume that whomever has the turn to speak may take the initiative. But assertions comprising information about higher-level goals, system proprietary knowledge, and virtually any other system beliefs can be used to pre-empt this default: for example, if time is running out and ACDM knows a likely route, it may interrupt. These assertions may result from completely independent inference streams in ACDM. Simultaneously, the system can reason about how well each of its goals are being addressed by the current conversation, note the time left to fulfill each of them, and analyze the speaker's last utterance to determine if any shift of turn or initiative is offered. Conclusions made in each of these streams may trigger preemption, though, of course, ultimately the system designer/ engineer must still encode the exact circumstances causing such a triggering.

We have argued earlier that ACDM's logical framework is a natural way to address the competencies needed for mixed-initiative dialogue. But perhaps the most basic of those competencies is communicative: the system must be able to make itself understood, even in the face of miscommunication. Competence, then, entails an ability to engage in sophisticated repair and clarification metadialogue.

6 Future directions

As described in the previous sections, the ACDM, whether viewed as an independent agent, or a module in a dialogue system, has many features of mixed initiative. The main feature that is still missing are full theories of control and decision-making. Active logic and the Alma/Carne provide the representational capacity for implementing different theories of who has the right to make decisions and control the interaction, but gives few guidelines for the best strategies for given tasks. The current ACDM takes very little initiative, like the TRAINS-96 dialogue manager that it replaces, usually following user requests, except in the case of problems that show up. The right amount of control is still a topic of current investigation. We hope to also be able to apply other results from similar systems, e.g., [Smith and Hipp, 1994].

6.1 Repair sub-dialogues

One focus of current work is an examination of how our emphasis on logical modeling of actions and execution to represent complex dialog contexts such as spontaneous repairs within a more extended repair sub-dialogue. For example, the system might utilize high-level rules which allow almost any complex action resulting in an system initiative to be interrupted by a sub-action `AddressUserRepair` that is instantiated upon the user initiating a repair. This sub-action, in turn, might be interrupted in appropriate circumstances by further repairs by either party. Thus clarification is seen as a potentially collaborative complex action which can be performed in many different ways, depending on who performs the initiatives needed to achieve the clarification.

6.2 Speech Challenges

Our current implementation of the dialogue system uses only a keyboard and mouse interface, which actually removes some of the mixed-initiative issues that appear in spoken dialog. Our future efforts are aimed at the inclusion of voice input, as well, with a focus on some of the issues of meta-reasoning about the input that may help overcome speech recognition problems.

The problems that we intend to deal with, in the context of using voice inputs are illustrated by the following examples.

```
Ex. 1: {MISSING SPEECH}
USER:  Have you sent the Boston train to Maryland?
SYS : [sends Boston train to Maryland]
USER: Don't do that now!
```

Due to problems in speech recognition, it is possible that the parser never gets the first part of the utterance. As a result, the system might end up doing something very different from what was expected. Now the user needs to be able to intervene immediately, and the system should be highly reactive, as if "Don't ... now!" is processed as a kind of logical panic button.

```
Ex. 2: {SENTENCE BOUNDARY}
USER: Send the Boston train to New York first send the
Maryland train to Boston.
```

It is not necessary that the period in written text always get translated directly to a pause in spoken text as illustrated by this example. Here, the word 'first' could belong either to the first phrase or to the second phrase. Punctuation will not be available to mark sentence boundaries, and prosodic processing disambiguate the various possibilities (e.g, as to which should be done first). In some cases, context might help; however, in some other cases, the system should take the initiative to direct a question to the user to help in disambiguation.

Acknowledgments

This work was supported in part by NSF grant IIS-9724937. This work is part of the dialogue effort by the University of Maryland Active Logic Group. We would also like to thank James Allen and George Ferguson from University of Rochester for allowing the use of the TRAINS-96 system as a platform in which to embed these ideas.

References

[Allen *et al.*, 1996] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. A robust system for natural spoken dialogue. In *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 62–70, 1996.

[Allwood *et al.*, 1989] Jens Allwood, Joakim Nivre, and Elisabeth Ahlsen. Speech management: On the non-written life of speech. Technical Report (GPTL) 58, Gothenburg Papers in Theoretical Linguistics, University of Göteborg, 1989.

[Clark and Schaefer, 1989] Herbert H. Clark and Edward F. Schaefer. Contributing to discourse. *Cognitive Science*, 13:259–294, 1989. Also appears as Chapter 5 in [Clark, 1992].

[Clark, 1992] Herbert H. Clark. *Arenas of Language Use*. University of Chicago Press, 1992.

[Cohen and Levesque, 1990] Phillip R. Cohen and Hector J. Levesque. Persistence, intention, and commitment. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Cohen and Levesque, 1991] Phillip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 35, 1991.

[Elgot-Drapkin and Perlis, 1990] J. Elgot-Drapkin and D. Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[Elgot-Drapkin, 1988] J. Elgot-Drapkin. *Step-logic: Reasoning Situated in Time*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1988.

[Elgot-Drapkin, 1991] J. Elgot-Drapkin. A real-time solution to the wise-men problem. In *Proceedings of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, Stanford, CA, 1991.

[Ferguson, 1995] George Ferguson. *Knowledge Representation and Reasoning for Mixed-Initiative Planning*. PhD thesis, University of Rochester, 1995. Also available as TR 562, Department of Computer Science, University of Rochester.

[Grosz and Sidner, 1990] Barbara J. Grosz and Candace L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Group, 1993] External Interfaces Working Group. Draft specification of the kqml agent-communication language. available through the WWW at: http://www.cs.umbc.edu/kqml/papers/, 1993.

[Gurney *et al.*, 1995] J. Gurney, D. Perlis, and K. Purang. Active logic and Heim's rule for updating discourse context. In *IJCAI 95 Workshop on Context in Natural Language*, 1995.

[Harel, 1979] D. Harel. *First Order Dynamic Logic*. Springer-Verlag, 1979.

[Kraus *et al.*, 1990] S. Kraus, M. Nirkhe, and D. Perlis. Deadline-coupled real-time planning. In *Proceedings of 1990 DARPA workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 100–108, San Diego, CA, 1990.

[Lochbaum, 1994] Karen E. Lochbaum. *Using Collaborative Plans to Model the Intentional Structure of Discourse*. Ph.d. dissertation, computer science department, Harvard University, Cambridge, MA, 1994.

[McRoy and Hirst, 1995] Susan W. McRoy and Graeme Hirst. The repair of speech act misunderstandings by abductive inference. *Computational Linguistics*, 21(4):5–478, 1995.

[Nirkhe *et al.*, 1997] M. Nirkhe, S. Kraus, M. Miller, and D. Perlis. How to (plan to) meet a deadline between *now* and *then*. *Journal of logic computation*, 7(1):109–156, 1997.

[Nirkhe, 1994] M. Nirkhe. *Time-situated reasoning within tight deadlines and realistic space and computation bounds*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1994.

[Perlis *et al.*, 1996] D. Perlis, J. Gurney, and K. Purang. Active logic applied to cancellation of gricean implicature. In *Working notes, AAAI 96 Spring Symposium on Computational Implicature*. AAAI, 1996.

[Sadek, 1992] M. D. Sadek. A study in the logic of intention. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 462–473, 1992.

[Smith and Hipp, 1994] Ronnie W. Smith and D. Richard Hipp. *Spoken Natural Language Dialog Systems: A Practical Approach*. Oxford University Press, 1994.

[Traum and Allen, 1994] David R. Traum and James F. Allen. Discourse obligations in dialogue processing. In *Proceedings of the 32^{nd} Annual Meeting of the Association for Computational Linguistics*, pages 1–8, 1994.

[Traum, 1994] David R. Traum. *A Computational Theory of Grounding in Natural Language Conversation*. PhD thesis, Department of Computer Science, University of Rochester, 1994. Also available as TR 545, Department of Computer Science, University of Rochester.