Comparing Agent Modeling for Language and Action

Nancy Green and Jill Fain Lehman
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
(Internet: nancy.green@cs.cmu.edu, jill.lehman@cs.cmu.edu)

Abstract

We present our approach to agent modeling for communication, and compare it to an approach to agent modeling for other types of action. The comparison should be instructive since both approaches are implemented in the same problem-solving architecture, face similar application domain requirements, and address the same general problem of comprehension. Also, we show how for discourse processing, it is possible to have the benefits of viewing agents in terms of their beliefs and intentions without sacrificing real-time performance and reactivity.

Introduction

Although solutions have grown up in very different research communities, the problem of agent modeling for communication bears striking similarity to the problem of agent modeling for planning and action. Thus, it would not be at all surprising if there were lessons to be shared between the two communities. In this paper, we present our approach to agent modeling for communication, and compare it to an approach to agent modeling for other types of action. Both approaches have been applied in the same domain and both have been implemented within the same integrated problem-solving and learning architecture, Soar.[1] First we describe agent modeling in NL-Soar, a computational model of real-time dialogue generation and comprehension which we have developed. NL-Soar's agent model represents the agents with which it communicates at the level of discourse actions. Next we briefly describe agent modeling in an automated pilot agent (AP) developed by another research group using Soar but addressing a different problem in the same domain: planning the real-time domain actions (non-communicative actions such as controlling an airplane) of an agent in a complex, dynamic, multi-agent domain (Tambe & Rosenbloom 1994). The AP's agent model represents the agents with which it interacts at the level of domain actions. In the rest of the introduction, we describe the problems shared by NL-Soar and AP in more detail.

[1] For more information on Soar in general, see (Newell 1990; Lehman, Laird, & Rosenbloom 1996). For more information on the domain in which the two approaches have been applied see (Laird *et al.* 1995; Tambe *et al.* 1995).

Shared Problems

Both the problem of agent modeling for communication and for planning for action can be viewed primarily as a comprehension problem. In other words, at a certain level of description the goal is to take a continuous stream of input and organize it into meaningful units that enable the agent to reason to an appropriate response. For communication the input is a stream of words[2], while for planning, it is the primitives of the action language. Then, the comprehension problem for language is to analyze the syntactic and semantic relations obtaining in the input, as well as to recognize the communicative intention underlying the speaker's use of the message in the current conversational context. (The problem of recognizing the communicative intention may require an analysis of discourse relations in the input.) Similarly, the problem for interpreting the domain (non-linguistic) actions of other agents is to analyze the structural relations among the actions in context to recover their domain intentions (e.g., another agent is turning in order to fly due east in order to evade me). Common issues faced in solving each type of comprehension problem include recognizing local ambiguity, recovering from failures in ambiguity resolution, and the inference of elements implicit in the input stream.

In the particular systems we want to compare there are two other common factors: architecture and domain. As mentioned above, both systems have been implemented in Soar, a production-system architecture with a ubiquitous learning mechanism that views behavior as the outcome of problem space search. Moreover, both approaches have been applied to the

[2] Of course, the actual input for a human listener would be a continuous acoustic signal, but for our purposes here, it suffices to consider the stream after segmentation into words. We make a similar abstraction in treating the input for planning agents to be the primitives of the action language.

same real-time domain of tactical air battle simulation. Thus, in addition to responding to the functional constraints of comprehension, both approaches have had to respond to constraints imposed by the architecture and by the real-time nature of the task. In the next two sections we explore how each approach tries to satisfy its constraints.

Agent Modeling in NL-Soar

This section presents our approach to agent modeling in NL-Soar. The first three subsections provide background on the requirements which NL-Soar is designed to meet, the overall design of NL-Soar, and how this design addresses these requirements. Next we describe discourse generation in NL-Soar, in particular, the role of an agent's beliefs and intentions in discourse planning. Finally, we describe how knowledge compiled during discourse planning is used to create NL-Soar's agent model for discourse comprehension.

Requirements

Our current research in NL-Soar is directed towards providing agents implemented in Soar with real-time dialogue generation and comprehension capabilities.[3] Furthermore it is assumed that non-linguistic tasks in the application domain of NL-Soar require a high degree of reactivity to ongoing, external events. That is, communication must not interfere with the Soar agent's ability to perform non-linguistic tasks in a timely manner. Thus, as we describe elsewhere (Lehman, Van Dyke, & Green 1995), the overall design of NL-Soar must satisfy three properties: linear complexity, interruptability, and atomicity. The first property, linear complexity, means that processing to understand or generate a message must take time that is roughly linear in the size of the message (necessary to keep pace with human rates of language use). The second property, interruptability, ensures that time-critical task behaviors cannot be shut out by language processing (and vice versa). The third property, atomicity, ensures that if language processing is interrupted, partially constructed representations are left in a consistent and resumable state.

Language Processing in NL-Soar

To understand how NL-Soar provides the desired communication model, we must first briefly review the components out of which Soar systems are organized. Figure 1 is a graphical representation of the portion of NL-Soar used for generating discourse-level behavior.

[3]Previous versions of NL-Soar addressed on-line parsing (Lewis 1993) and sentence generation (Rubinoff & Lehman 1994). In the version of NL-Soar described here, inputs and outputs consist of written words. Interfacing NL-Soar to a speech recognizer is an area of current research. Other areas of current research include using NL-Soar in modeling second language acquisition and simultaneous translation.

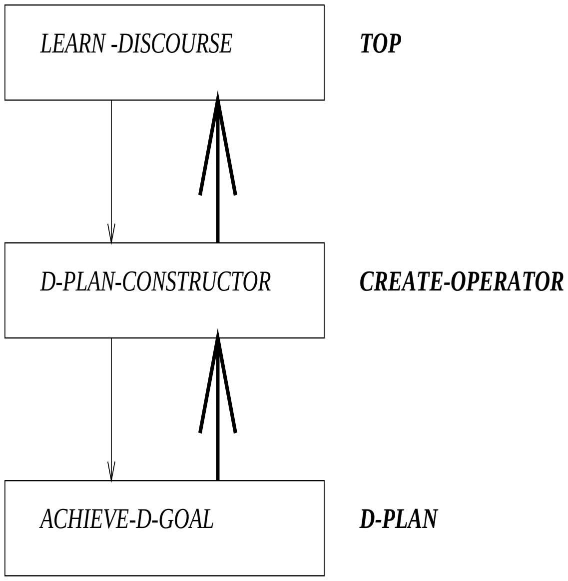

Figure 1: A goal stack for reasoning about language generation at the discourse level

Linguistic processes, like all processes in Soar, are cast as sequences of *operators* (names inside the boxes) that transform *states* (boxes) until a goal state is achieved. States and operators are further organized into *problem spaces* (boldface names). (Operators are defined by sets of productions stored in memory.) Comprehension problem spaces in NL-Soar contain operators that use input from the perceptual system to build syntactic, semantic and discourse structures on the state; generation problem spaces contain operators that use semantic and discourse structures to produce syntactic structures and motor output. Note that the problem space labelled Top is the only space connected to the perceptual and motor systems and the only space provided by the Soar architecture; all other problem spaces are provided by the system designer.

Behavior of the agent corresponds, in general, to the application of an operator to change the state. If the state changes caused by the operator, or by new inputs from the agent's sensors, satisfy the operator's termination conditions, then the operator is terminated and a new operator is applied. If the termination conditions remain unsatisfied, then a subgoal is created (thin arrows) and a new problem space is installed to make progress on solving the subgoal; in this way a goal/subgoal hierarchy may be generated. Processing in a subgoal often produces results to be returned to a state higher in the goal stack. When Soar's learning mechanism is being used, the return of results automatically creates *chunks*. Chunks are new productions added to production memory that capture the work done in the subspace, making it available in the superspace without subgoaling during future processing.

This means that when a system structured as in Figure 1 is *fully chunked* all of its behavior will be produced by operators in the Top space. We call these *top operators* and the behavior produced by them we call *top-level behavior*.

The top-level behavior of NL-Soar arises through learning. Figure 1 depicts the process by which one particular type of top language operator is learned, specifically, the d-plan-constructor. When a situation occurs in which there is a communicative goal for which no d-plan-constructor has previously been learned, a subgoal arises to build a d-plan-constructor. Various types of discourse knowledge (e.g., knowledge of how to achieve illocutionary goals, knowledge of turn-taking and grounding conventions) are brought to bear during the search for a sequence of appropriate discourse moves. When the moves have been constructed in the lower spaces, a d-plan-constructor is returned as a set of chunks. The application of this operator in the Top space will produce the planned set of discourse moves (to be realized by tactical language generation operators) and update the model of the discourse. Thus, during learning, NL-Soar uses Soar's goal stack to search for a new top operator that will compile together the results of a potentially lengthy search. Post-learning, that operator will, under similar conditions, be directly available for application in the Top space, and its changes to the state will be made without subgoaling. Note that while there are no theoretical limits on how much knowledge can be integrated through processing in the lower spaces, once a top operator has been constructed, the amount of processing it does is fixed.

The process we have just described for the creation of d-plan-constructors permeates all the linguistic levels and capabilities of NL-Soar. In other words, NL-Soar uses a similar dynamic to create top-level u-constructors (which build syntactic structure during comprehension), top-level s-constructors (which build semantic structure during comprehension), top-level d-realizers (which build structures during tactical generation), etc. When the linguistic operators required to comprehend or generate a sentence have been learned, language processing in NL-Soar looks like a succession of top operators of the different sorts modifying linguistic structures on the Top state on a more or less word-by-word basis. When a top operator has not been constructed via previous problem solving, the system automatically falls back on the knowledge in its lower spaces to produce behavior in a more deliberate fashion. Since the result of this deliberation is always the creation of a new top operator, however, the performance of the system as it processes more and more language becomes increasingly dominated by the more efficient form of that knowledge found in the Top space.

Linearity, Interruptability, and Atomicity

In real-time domains we are particularly concerned with the top-level behavior of the system, i.e. we intend to model dialogue among "domain experts" who have considerable prior experience within a domain. Top space language operators compiling syntactic and semantic knowledge are acquired by training NL-Soar to interpret or generate sentences exemplifying representative syntactic and semantic constructions. Note that it is not necessary for NL-Soar to be trained on exactly the same set of sentences that it will encounter post-training. Language operators compiling discourse knowledge are acquired by training NL-Soar to produce utterances while participating in training dialogues which exemplify representative discourse goals. Then, in an actual (as opposed to a training) dialogue, an agent's dialogue processing consists of a sequence of applications of top language operators possibly interleaved with non-linguistic operators.

How does a model trained in this way meet the requirements of linear complexity, interruptability, and atomicity? Once behavior is fully chunked, the arrival of a message results in the application of only a small number of top operators per word, the linear complexity we were after. Equally important, the language process itself is now represented in the Top space in terms of finely-grained operators that create the opportunity for interleavability. Since the application of an operator is defined by the Soar architecture as the non-interruptable unit of work, this means that the language process as a whole becomes interruptable at the level of work done by each top operator. This feature of the architecture gives us a certain degree of atomicity as well. Since the application of an operator is guaranteed to complete, all of its changes to the state will be executed before language is interrupted, leaving language processing in a resumable state.[4]

Discourse Generation in NL-Soar

The explicit representation of an agent's beliefs and intentions has been a dominant theme in computational approaches to both discourse generation and discourse interpretation.[5] In this field, plans are generally taken to be an agent's beliefs regarding his program of action towards accomplishing a particular goal (Pollack 1990). Furthermore, it is argued that providing an appropriate conversational response to a speaker's communicative act depends upon recognizing its role in the plans which the speaker intends for the hearer to recognize (Sidner 1985; Grosz & Sidner 1990).

Many plan-based approaches to discourse generation and comprehension have relied upon some amount of

[4]The ability to resume is limited by the fact that some Top state language structures are ephemeral, however. For example, the input buffer has a decay rate normally set to 2 seconds/word.

[5]For a survey of this field, see for example (Carberry 1990; Cohen, Morgan, & Pollack 1990).

compiled, declarative planning knowledge (sometimes referred to as *discourse recipes* (Pollack 1990)) provided by the system builder. An important distinction (Carberry 1990) is made between *domain goals*, goals to bring about changes in the world, versus *discourse goals*, goals to bring about changes in the beliefs or intentions of the hearer. In particular, typically, discourse plans are intended to update the conversational record or common ground, that is, a record of the changing mutual beliefs of the participants throughout the progress of a conversation (Thomason 1990). Necessarily, participants cannot really share beliefs, but much conversational effort is aimed at attempting to keep each participant's version of the record consistent with the other participants' apparent versions. Also, note that the distinction is sometimes made between an agent's apparent acceptance of a belief for the conversational record, as opposed to his actual adoption of the belief.

Residing in memory, NL-Soar's conversational record maintains information such as the current set of discourse expectations (e.g., the asking of a question creates the shared expectation that the addressee of the question will attempt to provide an answer), the current discourse segment purpose (i.e. the accepted goal of the current discourse segment (Grosz & Sidner 1986), which may span several turns), and the status of proposals that have been made in the preceding discourse (e.g., that a proposal has been understood and whether it has been accepted or challenged). In addition, NL-Soar represents in memory an agent's own beliefs which are relevant currently to the conversation (e.g., the information the agent may provide as an answer to a just-asked question). Finally, NL-Soar may represent in memory an agent's discourse goal (such as the goal that it become mutually believed that the participants have identified the same entity), which leads to the initiation of a discourse segment by the agent.

As described in the section on Language Processing in NL-Soar, the result of discourse planning in NL-Soar is a d-plan-constructor, a learned top operator. Note that a d-plan-constructor specifies conditions under which it is appropriate for it to be selected, in terms of the current state of the agent's conversational record, his other beliefs, and/or his discourse goals. Executing a d-plan-constructor fills a buffer with one or more discourse moves[6] to be realized by tactical generation operators and updates the conversational record to reflect the expected state of the conversation after the moves have been realized. In NL-Soar, an agent can be said to be commited to a discourse plan just in case the operator currently selected to run in the Top state is a d-plan-constructor. Note that in this view, the agent may hold a succession of plans aimed at satisfying the same discourse segment purpose.

D-plan-constructors are similar to the compiled planning knowledge (discourse recipes) used in traditional discourse generation systems, but with several significant differences:

- They are acquired by NL-Soar, during training, as a result of problem-solving to find coherent sequences of discourse moves to achieve discourse goals, rather than provided by the system builder.
- In some cases, the knowledge may be more specific than that typically provided in hand-coded discourse recipe libraries. In other words, although d-plan-constructors may be parameterized, they may also contain attribute-values which reflect the circumstances in which they were learned.
- Although the problem-solving process resulting in the acquisition of a d-plan-constructor may have involved decomposition into subgoals, the subgoal hierarchy is not retained in the d-plan-constructor. (Although information about hierarchical and enablement relations between actions is not saved in the current implementation, in principle, it could be encoded in the chunks for later use.)
- Whereas traditional discourse recipes are declarative specifications which must be interpreted at run-time, d-plan-constructors (like other operators in Soar) represent procedural knowledge.
- Most traditional approaches to discourse processing (one exception is (Green & Carberry 1994)) have not used the same discourse recipes for both generation and interpretation. In the next section, we describe how in NL-Soar d-plan-constructors are used in discourse interpretation.

The Agent Model in Discourse Comprehension

This section illustrates how the mechanisms of NL-Soar have influenced the form and function of NL-Soar's agent model for discourse comprehension. (For more details on our approach to discourse processing, see (Green & Lehman 1996).) Before describing the agent model, we describe the purpose of the agent model. The task of discourse comprehension in NL-Soar is to update the hearer's version of the conversational record in real-time,[7] based upon the current state of the conversational record and the hearer's semantic interpretation of the speaker's utterances. For example, given the semantic interpretation of an utterance as a name referring to the agent hearing the utterance, and given an appropriate discourse context, discourse comprehension recognizes that the name is being used as a summons,[8] and that the intended effect is to update the conversational record with the mutual belief

[6] In the current implementation, discourse planning is performed for one full turn.

[7] Our goal is to model discourse comprehension incrementally on a word by word basis. In our current implementation, discourse comprehension is not attempted until an utterance has been semantically interpreted as a whole.

[8] As opposed to a response to a wh-question, or an elliptical self-introduction.

of speaker and hearer that the speaking agent has begun a new turn and that the hearing agent will be the addressee of the turn.

Thus, one way of looking at discourse comprehension is as a problem of knowing how to update one's version of the conversational record appropriately. Also, as mentioned earlier, communication requires the recognition of the speaker's plans which the speaker intended for the hearer to recognize. Both problems could be solved if the hearer could recognize what d-plan-constructor had generated the speaker's discourse move. (The d-plan-constructor would specify any intended updates of the conversational record, enabling the hearer to update his own version accordingly.) We assume that within a given domain of expertise, agents do possess comparable sets of d-plan-constructors and further, intend for the application of those d-plan-constructors to be recognized.

In order to recognize the speaker's d-plan-constructor, first, it is necessary to create an agent model of the speaker, which we refer to as the hearer's model of the speaker (HMOS). The HMOS is constructed each time the top discourse comprehension operator is applied. Note that construction of the HMOS is simplified by the existence of the conversational record. That is, since the conversational record is presumed to normally consist of shared information, the HMOS version of the conversational record (that is, the hearer's model of the speaker's version of the conversational record) can be created by copying the hearer's version of the conversational record. Heuristics are used to supply non-shared information.

Then, in the context of HMOS, a d-plan-constructor is selected and applied by Soar, resulting in updates to HMOS. This d-plan-constructor models the operator which resulted in the speaker's production of the current discourse move. The hearer compares the predicted discourse moves to what he has heard (semantically interpreted) so far. If the current discourse move has been predicted by the d-plan-constructor, then the interpreting agent may update his version of the conversational record from the HMOS conversational record. Note that a d-plan-constructor may generate multiple discourse moves. Since discourse comprehension begins as soon as semantic operators have provided an interpretation for the current utterance, the HMOS may contain discourse moves which the speaker plans to make but which have not yet been heard. These moves guide the hearer's interpretation of subsequent utterances. Note that as soon as the discourse comprehension operator terminates, the agent model (HMOS) is deleted.

We have not yet addressed strategies for handling ambiguity and error recovery in discourse comprehension, although we envision using a general strategy consistent with that currently used by NL-Soar at the sentential level. The system keeps only a single syntactic and semantic interpretation for each utterance. Ambiguity arises when more than one top-level syntactic or semantic operator can be applied in the current context. Ambiguity resolution is accomplished using other knowledge sources to choose among the available operators. So, for example, two different syntactically appropriate attachments for a prepositional phrase might be decided by semantic knowledge that prefers one of them in the current context. If an incorrect attachment is revealed in the disambiguating region of the utterance, error recovery is required. Because only one path has been kept, standard backtracking is not possible. Instead, the same detection of local structural patterns that signals an error also leads to learning new top-level constructors that replace the previous structure with the appropriate structure. Since detection of errors is based on specific local structural patterns, not all errors can be undone; this leads, naturally, to a theory of unrecoverable errors which conforms to data on human on-line sentence comprehension (Lewis 1993). It is an interesting open question whether there are comparable limitations on human recognition of discourse intention.

Agent Modeling in AP

In this section, we briefly describe a different approach to agent modeling developed for the automated pilot agent described in the introduction. We compare it to NL-Soar's approach in hopes of gaining deeper insight into agent modeling for discourse.

Requirements

The automated pilot agent (AP) developed by Tambe and Rosenbloom (Tambe & Rosenbloom 1994) needs to monitor the occurrence of and temporal relationships between events in the world, a process which they refer to as *event tracking*. The events may be high-level or low-level actions, observed or simply inferred, and instigated by other agents or not. They define tracking as the process of recording the events and their temporal relationships, and monitoring their progress. According to Tambe and Rosenbloom, the problem of *event tracking* is closely related to the problem of plan recognition. They claim that within the domain of tracking opponents in simulated air combat, there are several novel issues faced in event tracking however. First, AP must track highly flexible and reactive behaviors of the other agents. That is, in this environment, it is not reasonable for an agent to plan far ahead and then commit to the plan. Second, the agents' actions continuously influence the actions of the other participants, thus interpretation of an agent's actions may require an understanding of these interactions. Third, event tracking must be performed in real-time and not interfere with the agent's ability to perform other actions in real-time. Last, event tracking must be able to continue efficiently in the presence of ambiguity.

The above issues are similar to some issues faced in discourse comprehension and generation by NL-Soar.

We claim that the first and second issues they raise are faced by dialogue participants too (although the need for flexibility and reactivity in their system is probably on a smaller timescale than in dialogue). In dialogue, speakers cannot commit themselves many moves in advance to a rigid discourse plan. A dialogue participant must monitor the reaction of the other participants to what he has just said, address misconceptions, follow changes in topic and interruptions, respond to unanticipated requests, etc. Also, as described earlier, one of NL-Soar's design goals is to model real-time language behavior. Lastly, the ability to continue discourse comprehension in the presence of ambiguity is an aspect of the general comprehension problem.

Design of AP

The automated pilot uses a problem space/operator goal stack to generate its real-time behavior. For example, AP may be attempting to execute its mission by applying the EXECUTE-MISSION operator in the top-level problem space. In the current situation, AP may not be able to satisfy this top-level goal, and so a subgoal is generated. In the subgoal problem space, an INTERCEPT operator is selected, which in turn may lead to a subgoal in which an EMPLOY-MISSILE operator is selected, and so on. In short, selection of an action at any level of abstraction is driven by the current situation. Tambe and Rosenbloom claim that this design supports the flexible and reactive behavior needed for this type of agent, since whenever current conditions change such that an operator's termination conditions are satisfied, the operator will be terminated and its subgoals, if any, will be deleted.

To briefly compare this approach to the approach taken in NL-Soar, note that the design of AP does not make use of Soar's learning mechanism. AP uses only the operators given to it by its system builders. In contrast, NL-Soar uses both operators which are learned (acquired by chunking) and which are given to it by its builders. Thus, during training, NL-Soar would make use of a problem space/operator goal stack to generate its behavior while compiling that problem solving into a set of language operators (discourse, syntactic, and semantic). However, in real-time, NL-Soar need not make use of this power. Instead, the learned language operators are applied in NL-Soar's top-level problem space, avoiding the need for potentially costly problem-solving in lower problem spaces.

Event Tracking in AP

The automated pilot agent's model of its opponent consists of an independent problem space/operator goal stack running concurrently with AP's own goal stack and which AP is able to monitor. This approach is based on the following assumptions:

- the opposing agent has a similar mechanism for generating its own behavior,
- the opposing agent has available the same set of operators as AP's, and
- AP is able to create an accurate model of the current situation as perceived by the opposing agent, which is referred to as $\text{state}_{opponent}$.

Thus, as $\text{state}_{opponent}$ changes, the opponent's simulated goal stack will change as described in the preceding section. In other words, given $\text{state}_{opponent}$, the opponent's goal stack, and the opponent's operators, AP can simulate the behavior of its opponent. Thus, AP can infer unobserved actions of the opponent, as well as hierarchical relationships between actions. Note that AP must corroborate its opponent's ongoing observed behavior with the predicted behavior.

In order for AP's simulation of its opponent to be correct, it is necessary for $\text{state}_{opponent}$ to be accurate enough to cause the goal stack to be an accurate representation of the opponent's actual goal stack. Note that this is similar to the problem faced in NL-Soar of generating the hearer's model of the speaker (HMOS). In AP, $\text{state}_{opponent}$ is constructed using three strategies:

- For certain features of the current situation, no distinction is made between the knowledge of AP and of his opponent. This knowledge is stored in a physically shared space referred to as a world-centered problem space.
- Not all details of $\text{state}_{opponent}$ are modeled.
- Some information, which may be incomplete or inaccurate, is supplied by an external agent (e.g., military intelligence reports).

AP's approach to ambiguity in event tracking, that is, cases in which more than one operator is applicable in $\text{state}_{opponent}$, is to use heuristics to commit to a single interpretation and proceed. If inconsistencies between predicted and observed actions are detected later, then single-state backtracking is performed to undo some of the commitments and repair the inconsistencies.

Commonalities and Differences

This section summarizes the differences and commonalities in the two approaches to agent modeling. First, agent modeling in AP makes use of the Soar architecture in a different way than agent modeling in NL-Soar does. AP's agent model exists as a separate goal stack running concurrently with AP's goal stack. Note that this approach is consistent with AP's use of the goal stack to generate its own flexible and reactive behavior in real-time. On the other hand, NL-Soar's real-time behavior is generated primarily by use of *learned* top operators. NL-Soar's agent model is a simulation of the other agent's top state (the Hearer's Model of the Speaker), based upon the current conversational record and updated by shared d-plan-constructors, i.e. learned discourse operators.

Another difference is that NL-Soar's discourse comprehension component models recognition of discourse plans, which are intended to be recognized, whereas the goal of event tracking in AP is to recognize actions which are not necessarily intended to be recognized. The d-plan-constructors used in NL-Soar can be viewed as constructing shared plans (Grosz & Sidner 1990). Thus, the problem of agent modeling in discourse comprehension may be addressed using a set of assumptions which are not applicable in the latter type of agent modeling: e.g., that the conversational record can be used to construct the HMOS, that the agents share a set of operators for generating the activity to be tracked, and that linguistic clues may be provided to assist comprehension. Note that in subsequent work (Tambe 1996), AP has been extended to track "team" activities, activities involving joint intentions and joint commitments to joint activities, using team operators. However, no distinction is made between the problem of tracking team activities by members and by non-members of the team, i.e., the distinction between intended and non-intended recognition.

One feature which the two approaches have in common is the general approach of constructing an initial model of the other agent and then simulating his behavior in order to recognize his underlying intentions. This similarity is not coincidental. Our approach to agent modeling has been influenced by the work on event tracking for AP. Moreover, while a more traditional approach to discourse plan recognition could have been implemented for NL-Soar, e.g., using plan-inference rules to reason from surface speech acts to the speaker's plans, the present approach capitalizes on the architectural features of Soar (chunking, control structure, etc.). Another point of similarity is the approach to ambiguity in AP and in sentence comprehension in NL-Soar. Both commit to a single interpretation and limit backtracking due to their goal of modeling real-time behavior.

Previous research in discourse has shown that discourse interpretation requires recognition of domain goals as well as discourse goals. An interesting open issue for NL-Soar is how to integrate recognition of the two types of goals. Should AP's approach to event tracking be used in recognizing domain goals for discourse comprehension in NL-Soar, or something else? Note that the two approaches are not mutually exclusive alternatives of the architecture. In AP's agent model, intentions are represented implicitly by the semantics of the goal stack used to simulate actions of the observed agent. In NL-Soar, a d-plan-constructor is the result of "compiling away" the discourse generation goal stack, resulting in a learned top-state operator that can be used to simulate the discourse actions of the observed agent. One way of integrating the two approaches would be to simulate another agent's observed behavior, linguistic and non-linguistic, with the same goal stack. Precompiled language operators, e.g., d-plan-constructors, would run in the top-level language state of this goal stack.

Summary

We have described our approach to agent modeling in NL-Soar, and compared it to the approach used in the automated pilot agent. The comparison should be instructive since both approaches are implemented in the same problem-solving architecture, face similar application domain requirements, and address the same general problem of comprehension. We showed that for discourse processing, it is possible to have the benefits of viewing agents in terms of their beliefs and intentions without sacrificing real-time performance and reactivity. Also, we showed that the differences in the two approaches to agent modeling are due in part to their different use of architectural mechanisms in the architecture which they share.

Acknowledgments

This research was supported in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U.S. government. We wish to thank Milind Tambe and an anonymous reviewer for their comments on the preliminary version of this paper.

References

Carberry, S. 1990. *Plan Recognition in Natural Language Dialogue.* Cambridge, Massachusetts: MIT Press.

Cohen, P.; Morgan, J.; and Pollack, M., eds. 1990. *Intentions in Communication.* Cambridge, Massachusetts: MIT Press.

Green, N., and Carberry, S. 1994. A hybrid reasoning model for indirect answers. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics.*

Green, N. L., and Lehman, J. F. 1996. An approach to compiling knowledge for dialogue generation and interpretation. Submitted for publication.

Grosz, B., and Sidner, C. 1986. Attention, intention, and the structure of discourse. *Computational Linguistics* 12(3):175–204.

Grosz, B., and Sidner, C. 1990. Plans for discourse. In Cohen, P.; Morgan, J.; and Pollack, M., eds., *Intentions in Communication.* MIT Press. 417–444.

Laird, J. E.; Johnson, W. L.; Jones, R. M.; Koss, F.; Lehman, J. F.; Nielsen, P. E.; Rosenbloom, P. S.; Rubinoff, R.; Schwamb, K.; Tambe, M.; Dyke, J. V.;

van Lent, M.; and III, R. E. W. 1995. Simulated intelligent forces for air: The Soar/IFOR project 1995. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation.*

Lehman, J. F.; Laird, J. E.; and Rosenbloom, P. S. 1996. A gentle introduction to Soar, an architecture for human cognition. In Sternberg, S., and Scarborough, D., eds., *Invitation to Cognitive Science*, volume 4. MIT Press.

Lehman, J. F.; Van Dyke, J.; and Green, N. 1995. Reactive natural language processing: Comprehension and generation in the air combat domain. In *Proceedings of the AAAI 1995 Fall Symposium on Embodied Action.*

Lewis, R. L. 1993. *An Architecturally-based Theory of Human Sentence Comprehension.* Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania. Available from Computer Science Department as Technical Report CMU-CS-93-226.

Newell, A. 1990. *Unified Theories of Cognition.* Cambridge, Massachusetts: Harvard University Press.

Pollack, M. 1990. Plans as complex mental attitudes. In Cohen, P.; Morgan, J.; and Pollack, M., eds., *Intentions in Communication.* MIT Press.

Rubinoff, R., and Lehman, J. F. 1994. Real-time natural language generation in NL-Soar. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, 199–206.

Sidner, C. L. 1985. Plan parsing for intended response recognition in discourse. *Computational Intelligence* 1:1–10.

Tambe, M., and Rosenbloom, P. S. 1994. Event tracking in a dynamic multi-agent environment. Technical Report ISI/RR-RR-393, Information Sciences Institute, University of Southern California, Marina del Rey, California.

Tambe, M.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine* 16(1):15–39.

Tambe, M. 1996. Tracking dynamic team activity. In *Proceedings of AAAI 1996.*

Thomason, R. H. 1990. Accommodation, meaning, and implicature: Interdisciplinary foundations for pragmatics. In Cohen, P.; Morgan, J.; and Pollack, M., eds., *Intentions in Communication.* Cambridge, Massachusetts: MIT Press. 325–363.