An Application of Explanation-Based Learning to Discourse Generation and Interpretation

Nancy Green and Jill Fain Lehman
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15231 USA
nancy.green@cs.cmu.edu

Abstract

This paper presents a brief description of Explanation-Based Learning (EBL), and argues that it is an approach to machine learning with significant potential for use in discourse processing. More specifically, EBL can be used by systems that model discourse generation as goal-driven behavior, and that model discourse interpretation as recognizing the speaker's discourse goals. As evidence, we describe the discourse architecture of the NL-Soar dialogue system. The architecture uses the EBL capabilities of Soar to compile discourse recipes during discourse planning. The recipes can be reused to speed up discourse generation and as a knowledge source during discourse interpretation.

Introduction

This paper presents a brief description of Explanation-Based Learning (EBL), and argues that it is an approach to machine learning with significant potential for use in discourse processing. More specifically, EBL can be used by systems that model discourse generation as goal-driven behavior, and that model discourse interpretation as recognizing the speaker's discourse goals. As evidence, we describe the discourse architecture of the NL-Soar dialogue system. The architecture uses the EBL capabilities of Soar to compile discourse recipes during discourse planning. The recipes can be reused to speed up discourse generation and as a knowledge source during discourse interpretation.

Explanation-Based Learning

Explanation-Based Learning (EBL) refers to an approach to machine learning where a program learns by formulating an *explanation* of why a single training example is an instance of the concept to be learned. This explanation is transformed into an operational description which can be used to identify new instances of the concept. Formulating such an explanation requires domain knowledge. In contrast to corpus-based approaches, EBL can make meaningful generalizations from a single example.

In one of the first articles to survey and attempt to unify previous work on explanation-based approaches, Mitchell et al. provide a definition of and method for constructing an *explanation-based generalization* (EBG) (Mitchell, Kellar, & Kedar-Cabelli 1986). Knowledge about the training example and domain is encoded as logical propositions and the process of finding an explanation is characterized as finding a proof that the training example satisfies a (non-operational) concept definition. The proof is converted into an operational definition that can be used to identify other instances of the concept. In other words, the explanation is used to find sufficient conditions for recognizing new instances of the concept.

Dejong and Mooney propose a different characterization of explanation-based approaches, which they term *Explanation-Based Learning* (EBL) (Dejong & Mooney 1986) to reflect a broader range of phenomena than considered by Mitchell et al. While Mitchell et al. characterize the approach in theorem-proving terms, Dejong and Mooney characterize it in problem-solving terms, i.e., in terms of initial states, goal states, operators, and plans to achieve the goal from the initial state. Also, they distinguish two types of EBL systems. In one, the explanation is derived solely by an internal problem-solver making use of its domain theory. While this approach is capable in theory of producing an optimal solution, it may be inefficient. Dejong and Mooney propose an alternative approach to EBL in which the system constructs an explanation using externally supplied "hints", which reduce the amount of search and which bias the system towards certain explanations. The hints are provided to the system as a sequence of observed states and actions, representing a partial explanation of how to achieve the goal.

To summarize the characterization of EBL given by Dejong and Mooney (extending the one given by Mitchell et al.), an EBL problem is said to consist of:

- a domain theory: a set of facts and inference rules about the domain and a library of primitive problem-solving operators and previously learned or hand-coded schemata (where a *schema* is defined as a partially-ordered set of operators for achieving a goal),

- the goal state,
- the initial world state, and
- (optionally) an observed sequence of primitive operators which specify (partially or completely) how to achieve an instance of the goal. (If no observations are supplied, then the system constructs a plan on its own to satisfy the goal.)

In the paper, they propose a domain-independent method for deriving the explanation, and illustrate its application to the problem of acquiring a schema for recognizing stories describing kidnapping events from a single kidnapping narrative. The domain theory provides schemata for actions described in kidnapping narratives, as well as general inference rules about this domain. The goal state is the goal of the particular kidnapping narrative provided as input, e.g., that the kidnapper will possess $250,000. The initial state and observed operator sequence is a story about a particular kidnapping event in the language of the domain theory. An explanation is derived and generalized from the story. The explanation is transformed into a schema with preconditions (conditions under which the schema is applicable) and effects.

An Integrated Discourse Architecture Using EBL

We have developed an integrated discourse architecture for the NL-Soar dialogue system (Lehman *et al.* in preparation). The goal of NL-Soar is to enable intelligent agents to communicate via natural language with humans in task-oriented dialogues in restricted domains. Currently, inputs and outputs consist of written natural language. The functional components of NL-Soar include sentence comprehension (parsing and semantic interpretation), sentence generation, discourse generation, and discourse comprehension. In the discourse architecture, discourse plan operators or "recipes" are compiled automatically during discourse planning using an EBL-style approach. In future situations, these discourse recipes are used to speed up discourse generation and as a knowledge source for discourse comprehension. For a full description of the discourse architecture, see (Green & Lehman in preparation). In this paper, its features are summarized and the use of EBL-style learning in it is explicated.

Overview of Learning in NL-Soar

NL-Soar is implemented using the Soar architecture (Laird & Rosenbloom 1990; Lehman, Laird, & Rosenbloom 1996; Newell 1990). Intended as an architecture for general intelligence, Soar provides a general learning mechanism, called *chunking*, that operates automatically during problem-solving. Problem-solving knowledge is encoded as productions (condition-action rules) in the system. As control returns from subgoals during problem-solving, Soar automatically creates chunks, i.e., new productions which compile the results of the problem-solving experience. The conditions of a chunk reflect the aspects of the state that were relevant to achieving the subgoal. The actions of a chunk reflect the changes to the state accomplished by the subgoal. (Both conditions and actions of chunks may contain variables.) In future situations where the conditions of a chunk hold, Soar can apply the chunk instead of performing the potentially costly problem-solving which gave rise to the chunk. Chunking can be regarded as a type of EBL, where the productions that were applied during subgoaling constitute the explanation (Rosenbloom & Laird 1986). The conversion of this explanation to chunks results in operational knowledge which can be applied to future problem-solving.

Although some aspects of the behavior of a system implemented in Soar are determined by the Soar architecture (e.g., how production rules are selected, automatic chunking), a system builder's organization of the problem-solving process contributes to the behavior of the system, including what types of chunks are acquired. In NL-Soar, the discourse planning process has been organized so that the resulting chunks will have properties useful for discourse generation and interpretation. (The organization of NL-Soar at the discourse-level closely follows that at the sentence-level. For information on NL-Soar's sentence-level comprehension and generation processes, see (Lehman *et al.* in preparation). Also, note that chunking has been put to a variety of other uses in other systems built in Soar, e.g., (Miller & Laird 1996; Miller, Lehman, & Koedinger submitted; Huffman 1993; Johnson, Krems, & Amra 1994).)

Discourse Architecture Components

As shown in Figure 1, the input to discourse generation in NL-Soar is the shared state of the dialogue, termed the *conversational record* (Thomason 1990), and the speaker's salient non-shared beliefs and desires. The output is a sequence of primitive discourse actions to be realized by the sentence generator, and their anticipated effects (e.g., updates to the conversational record). In NL-Soar, discourse generation is modeled as goal-driven behavior. During discourse planning, however, chunking results in the automatic creation of discourse recipes as a side-effect of planning. Thus, planning is integrated with learning. In future situations, whenever a learned discourse recipe is applicable, it is used instead of planning. To describe this process in EBL terms, the domain theory consists of knowledge of primitive discourse actions and their preconditions and effects, and the planning algorithm. The goal state is the desired state of the dialogue, and the initial state is the current state of the dialogue. The plan provides an *explanation* of how these effects can be achieved. The explanation is then generalized to provide a dis-

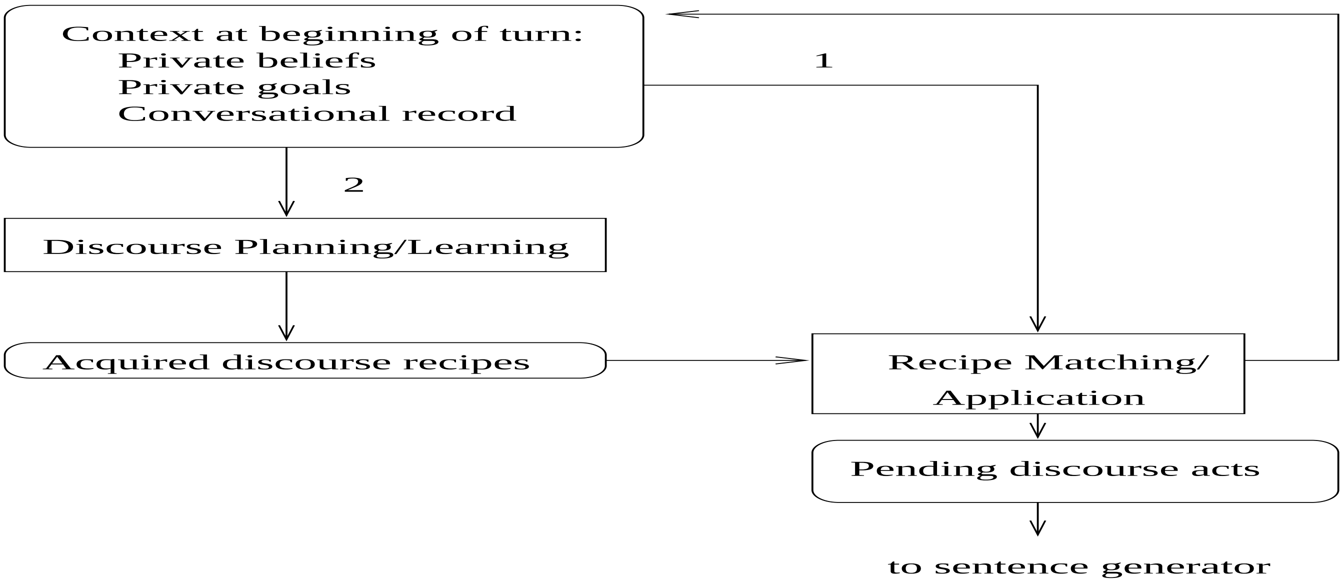

Figure 1: Discourse Generation Architecture

course recipe, describing an abstract discourse action and its preconditions and effects.

In our implementation, discourse generation is triggered whenever the agent is not engaged in a higher-priority task, the dialogue turn is available, and either the agent has a private desire to communicate or there is a discourse expectation (according to the conversational record) for the agent to communicate. The flow of control through the discourse generation components (shown as rectangles) in the figure is as follows. First, (taking the arc labelled 1) the Recipe Matching component applies any discourse recipe which is applicable in the current context. (Currently, we are using Soar's operator selection mechanism to match the appropriate recipe. Since the recipes are designed to apply in mutually exclusive contexts, no conflict resolution procedure is required.) Applying a recipe updates the context, and adds a set of pending primitive acts to the Sentence Generator's input buffer. If no recipe is applied, (taking the arc labelled 2) the Discourse Planning/Learning component creates a recipe, which is added to the store of acquired recipes, then matched by the Recipe Matching component, and applied. Planning is performed for a single turn at a time. Thus, each acquired recipe characterizes a set of similar turns, i.e., a set consisting of all possible instantiations of the recipe's variables.

Use of a recipe can speed up discourse generation, since no planning is required. In addition to using the recipes to speed up generation, NL-Soar uses the recipes that were acquired during discourse generation for discourse comprehension. As shown in Figure 2, the input to discourse comprehension is the syntactic and semantic interpretation of an utterance and the conversational record. Recognizing the speaker's meaning is modeled as recognizing which recipe must have given rise to the observed utterance and was intended to be so recognized, in virtue of the agents sharing similar processing mechanisms, similar sets of recipes (acquired through similar experience), and the conversational record. The output is the set of updates to the conversational record specified by the instantiated recipe that was recognized.

In our implementation of discourse plan recognition, the speaker's discourse generation process is simulated by the hearer and compared to what is actually heard. In brief, the conversational record is used to construct part of a model of the speaker, which we refer to as the Hearer's Model of the Speaker (HMOS). The Recipe Matching component used in discourse generation is used to select the discourse recipe which is applicable in the context of the HMOS. The selected recipe is applied, resulting in updates to the HMOS. The hearer's Monitor component compares updates to the HMOS to what has actually been heard. If the simulated updates to the HMOS are consistent with what has been heard, the Monitor updates the hearer's version of the conversational record. We decided to investigate this approach to plan recognition, which is similar to earlier work on event tracking in Soar (Tambe & Rosenbloom 1995) (but which is quite different from most prior approaches to discourse plan recognition (Carberry 1990)) because it permits much of the processing to be performed automatically by the underlying Soar architecture.

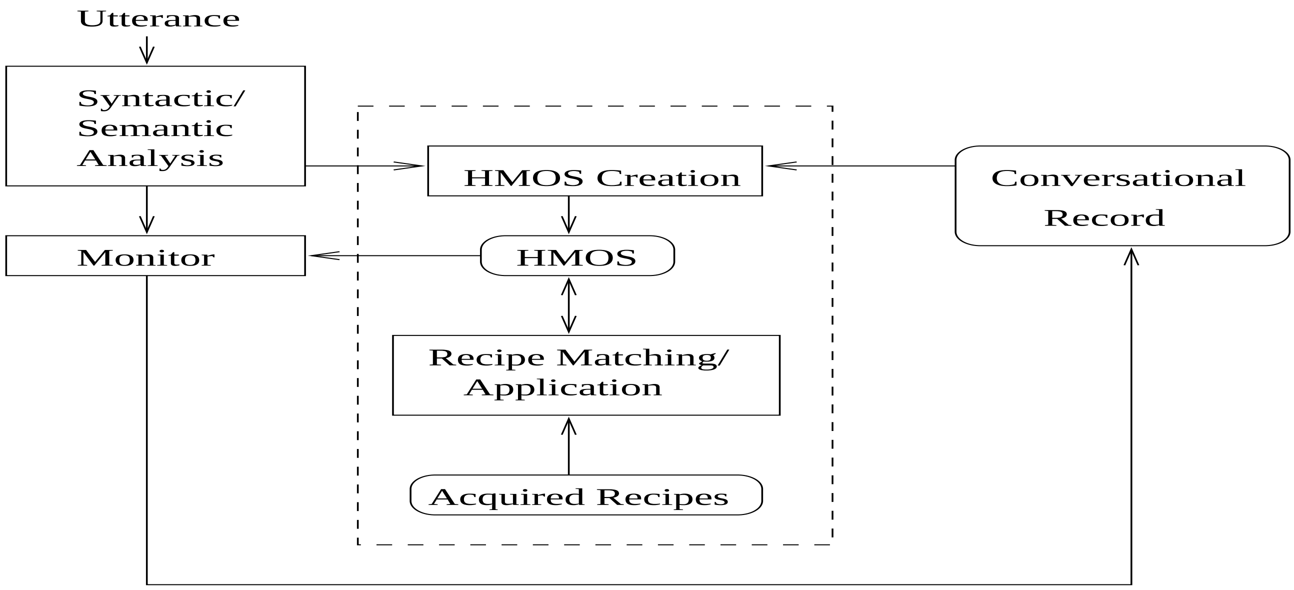

Figure 2: Discourse Comprehension Architecture

Discourse Recipes

NL-Soar acquires recipes by use of Soar's built-in chunking mechanism. Because of how the discourse generation component of NL-Soar's production system is organized, all recipes have a similar form. Each recipe consists of a set of chunks (learned condition-action rules). One of the chunks, the *proposal* chunk, specifies when the recipe is applicable for generation. The conditions of this chunk specify a generalization of the conditions that were tested at the onset of planning, e.g., particular features of the conversational record, and, possibly, salient non-shared beliefs and/or goals of the speaker. The action of this chunk is to allow the rest of the chunks (called *implementation chunks*) of the recipe to fire. The implementation chunks perform updates to the conversational record and put primitive discourse actions into the sentence generator's input buffer.

Application Domain

NL-Soar's discourse component has been implemented for the air-combat training domain, based on an analysis of transcripts of 20 dialogues. In the transcripts, human speakers are communicating over a (simulated) two-way radio channel while engaging in simulated air-combat exercises. The goal of the current implementation is for NL-Soar to enable a human trainee to communicate with an automated pilot agent playing the role of the other participant. As in other domains involving two-way radio discourse (Gibbon 1985), a large portion of each turn in the dialogues is devoted to discourse acts for managing the turn (Sacks, Schegloff, & Jefferson 1974) (*summons*, *self-introduction*, *end-turn*) and grounding (Clark &

	UTTERANCE	DISCOURSE ACT
A2:	Parrot101.	summons
	This is Parrot102.	self-introduction
	I have a contact bearing 260.	inform-description-of
	Over.	end-turn
A1:	Parrot102.	summons
	This is Parrot101.	self-introduction
	Roger.	acknowledge
	I have a contact bearing 270.	inform-description-of
	Over.	end-turn
A2:	Parrot101.	summons
	This is Parrot102.	self-introduction
	Roger.	acknowledge
	That is your bogey.	inform-same-object
	Over.	end-turn

Figure 3: Sample Dialogue

Schaefer 1989) (*acknowledge*). Most of the task-related dialogue can be described using three discourse segment purposes (DSPs) (Grosz & Sidner 1986). Seven primitive illocutionary act types and a small set of discourse expectations are sufficient to describe those dialogue segments.

For example, Figure 3 shows a dialogue segment constructed to illustrate a variety of discourse acts in this domain. The agents/speakers labeled A1 and A2 are known by the radio call-signs Parrot101 and Parrot102, respectively. The figure illustrates one of the various possible sequences of turns in a segment with a DSP described as *mb-describe-same-object(x)* (where x is a variable for object type), which can be glossed as the goal that the participants have the mutual belief that they have identified the same object (of type x). The *inform-description-of(x)* act which is used to initiate the segment in turn one and as a response in turn two, can be glossed as the act of providing a description of an object of type x. In the third turn, A2 proclaims that the DSP has been achieved by performing an *inform-same-object(x)* act.

Example Recipe

Currently in NL-Soar, planning is performed for a single turn at a time. Thus, each acquired recipe characterizes a set of similar turns, i.e., a set consisting of all possible instantiations of the variables of the recipe. To give an example of a discourse recipe, let us consider the one created during the planning of turn 1 of Figure 3.

The left-hand side of the proposal chunk for this recipe can be paraphrased as:

- according to the conversational record, there is currently no DSP, the medium is two-way radio, and the turn is available,
- the agent's/speaker's private beliefs include a belief *can-describe-object(x)*, which can be glossed as *the agent is certain that he can describe an object of type x*,
- the agent/speaker has a private goal, *mb-describe-same-object(x)*.

When the chunk is instantiated to generate turn one, x is instantiated with the type of object, *contact type*. The description of the particular object is constructed outside of the discourse component. The right-hand side of the proposal chunk enables the recipe's implementation chunks to fire in the current state.

The implementation chunks specify the following actions:[1]

- add the following acts to the sentence generation input buffer: *summons*, *self-introduction*, *inform-description-of(x)*, and *end-turn*,
- record a DSP of *mb-describe-same-object(x)* in the conversational record, and
- record the dialogue proposal made by means of the *inform-description-of(x)* act to the conversational record.

When the chunks are instantiated to generate turn one of the sample dialogue, x is instantiated with the type of object, *contact type*. In addition to the updates to the conversational record specified in the recipe, the *inform-description-of(x)* act will trigger updates to the set of current discourse expectations in the conversational record.

To summarize the differences between the recipes acquired during chunking by NL-Soar and hand-coded recipes used in traditional systems:

- They are designed for use in both generation and comprehension.
- Whereas a traditional discourse recipe (as described in, e.g., (Cawsey 1992; Hovy 1988; Lambert & Carberry 1991; Litman & Allen 1987; Maybury 1992; Moore 1995)) is a declarative specification, which must be interpreted at run-time, these recipes are encoded as sets of productions. The learned productions are incrementally acquired with experience, are in the same representation scheme as hand-coded productions, and are applied by the problem-solving architecture just as hand-coded productions are.
- The discourse architecture enables the system to fall back upon planning whenever the system has no recipe applicable to the current situation.

Discourse Planning

The current planner uses a very simple forward-chaining algorithm (implemented in about 35 production rules), which plans one turn at a time. A turn is planned in three stages: (1) acts to open the turn, (2) illocutionary acts, and finally (3) acts to close the turn. Acts for opening and closing the turn are chosen to perform turn-taking and grounding functions, independently of the DSP. Preconditions of the primitive illocutionary acts refer to the conversational record and/or non-shared salient beliefs/desires. For example, an *inform-description-of(x)* act (as illustrated in turn two of the sample dialogue) is licensed by the discourse expectation of an *inform-description-of(x)* act (in the set of current discourse expectations, which were evoked by the *inform-description-of(x)* act of turn one), the DSP *mb-describe-same-object(x)*, and by the agent's private belief *can-describe-object(x)*.

Training and Evaluation

Discourse recipes are acquired during training sessions in order to provide real-time performance during deployment of NL-Soar. (However, except for the time required for planning, there is no reason why recipe acquisition cannot be performed during deployment.

[1] The implementation chunks fire in parallel. Ordering constraints on the dialogue acts are represented declaratively for use by the sentence generator.

NL-Soar automatically falls back upon discourse planning whenever it has no applicable discourse recipe.) During training, NL-Soar is used to generate the contributions of both participants in representative dialogues. The contributions are elicited by an interactive program which enables the system builder to add the speaker's private beliefs and desires to working memory. (During deployment, these beliefs and desires are generated by the automated pilot agent.) Discourse recipes are added to production memory by Soar as a by-product of discourse planning during these sessions. Note that only a single example of a class of turns is required for each recipe. The recipes, which are in the same representational format as hand-coded productions, can be examined (and even modified) by the system builder if desired. They can be saved and reloaded with hand-coded productions at a later time.

Eight recipes (each describing a generalized turn) have been acquired and are sufficient to generate not only most aspects of the dialogues occurring in our corpus, but all possible dialogues which can be constructed by sequencing turns generated by the recipes. We expect recipes to speedup discourse generation in principle, since the number of serial operations required to select and apply a recipe is fewer than the number required during planning. However, we have not made performance measurements since the speedup to be gained in our current implementation, which uses a very simple planner, is not likely to be compelling. Another way of evaluating the recipes, which we have performed, was to test them in discourse comprehension. All of the recipes have been so used successfully.

Comments

We have not yet attempted to model dialogues with nesting, such as clarification subdialogues. Also, the planning algorithm is simple and does not address issues such as determining whether multiple goals can be achieved by the same act, or whether the achievement of one goal interferes with the achievement of another goal. However, our implementation demonstrates the potential of an architecture integrating discourse planning and explanation-based learning. Also, although Soar is not required in order to implement such an architecture, we made use of it for two reasons. First, we wanted to see if the organization of problem-solving/learning used in NL-Soar's syntactic and semantic processing components, could be adapted to discourse processing. More generally, Soar is an experimental platform for the development of computational models of intelligence. Thus, it is interesting to see how this platform constrains the discourse architectures that can be developed.

Other uses of EBL might be incorporated into a discourse architecture such as this. One possibility is to use EBL during discourse planning to acquire search control knowledge rather than recipes. This search control knowledge would be used by the discourse planner in the future to make discourse planning more efficient. For example in PRODIGY, EBL is integrated with a means-end problem-solver to acquire domain-specific search control rules to improve future problem-solving performance in domains such as machine-shop scheduling (Minton *et al.* 1989). These search control rules help select operators during the planning process reducing the amount of search performed by the planner. Similarly, during discourse planning search control rules could be learned to avoid exploring discourse strategies which would be ineffective.

Related Work

We are not aware of prior work to perform recipe acquisition during discourse planning. There has been some work, however, on using discourse recipes acquired automatically from a manually-annotated corpus (Alexandersson, Reithinger, & Maier 1997) for discourse plan recognition.

Although EBL has been applied to optimizing performance during sentence parsing (Neumann 1997) and sentence generation (Samuelsson & Rayner 1991), there has been very little work in applying it to problems in discourse. EBL has been used to learn rules for interpreting indirect speech acts (Schulenburg & Pazzani 1989). Given a representation of the surface speech act of an utterance used as an indirect speech act, the system uses the plan inference technique of (Allen & Perrault 1980) to interpret the utterance. EBL techniques are then used to compile the chain of inference into rules for use in future cases. Also, as described above, EBL has been used to recognize narrative schemata (Dejong & Mooney 1986).

Conclusions

We argue that EBL has potential for use in dialogue systems that model discourse generation as goal-driven behavior, and that model discourse interpretation as recognizing the speaker's discourse goals. As evidence, we describe the discourse architecture of NL-Soar. The architecture uses the EBL capabilities of Soar to compile discourse recipes during discourse planning. The recipes can be used to speed up discourse generation and as a knowledge source during interpretation.

Acknowledgments

Research on NL-Soar 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.

References

Alexandersson, J.; Reithinger, N.; and Maier, E. 1997. Insights into the dialogue processing of VERBMOBIL. In *Proceedings of ANLP-97.*

Allen, J. F., and Perrault, C. R. 1980. Analyzing intention in utterances. *Artificial Intelligence* 15:143–178.

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

Cawsey, A. 1992. *Explanation and Interaction: The Computer Generation of Explanatory Dialogues.* Cambridge, Massachusetts: MIT Press.

Clark, H. H., and Schaefer, E. F. 1989. Contributing to discourse. *Cognitive Science* 13:259–294.

Dejong, G., and Mooney, R. 1986. Explanation-based generalization: An alternative view. *Machine Learning* 1:145–176.

Gibbon, D. 1985. Context and variation in two-way radio discourse. *Discourse Processes* 8:395–419.

Green, N., and Lehman, J. F. in preparation. An integrated architecture for discourse: Generation, interpretation and recipe acquisition.

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

Hovy, E. H. 1988. Planning coherent multisentential text. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, 163–169.

Huffman, S. B. 1993. *Instructable Autonomous Agents*. Ph.D. Dissertation, University of Michigan.

Johnson, T. R.; Krems, J.; and Amra, N. K. 1994. A computational model of human abductive skill and its acquisition. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 463–468. Lawrence Erlbaum Associates.

Laird, J. E., and Rosenbloom, P. S. 1990. Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of AAAI-90.*

Lambert, L., and Carberry, S. 1991. A tripartite plan-based model of dialogue. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 47–54.

Lehman, J. F.; Dyke, J. V.; Lonsdale, D.; Green, N.; and Smith, M. in preparation. A building block approach to a unified language capability.

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.

Litman, D., and Allen, J. 1987. A plan recognition model for subdialogues in conversation. *Cognitive Science* 11:163–200.

Maybury, M. T. 1992. Communicative acts for explanation generation. *International Journal of Man-Machine Studies* 37:135–172.

Miller, C. S., and Laird, J. E. 1996. Accounting for graded performance within a discrete search framework. *Cognitive Science* 4(1):499–537.

Miller, C.; Lehman, J. F.; and Koedinger, K. submitted. Goal-directed learning in microworld interaction. *Cognitive Science.*

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

Mitchell, T. M.; Kellar, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1:47–80.

Moore, J. D. 1995. *Participating in Explanatory Dialogues.* MIT Press.

Neumann, G. 1997. Applying explanation-based learning to control and speeding-up natural language generation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, 214–221.

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

Rosenbloom, P. S., and Laird, J. E. 1986. Mapping explanation-based generalization onto Soar. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 561–567.

Sacks, H.; Schegloff, E. A.; and Jefferson, G. 1974. A simplest systematics for the organization of turntaking for conversation. *Language* 50:696–735.

Samuelsson, C., and Rayner, M. 1991. Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings*, 609–615. International Joint Conference on Artificial Intelligence.

Schulenburg, D., and Pazzani, M. J. 1989. Explanation-based learning of indirect speech act rules. Technical Report Technical Report 89-11, Dept. of Information and Computer Science, University of California, Irvine.

Tambe, M., and Rosenbloom, P. S. 1995. RESC: An approach for real-time, dynamic agent tracking. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.*

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.