Compiling Knowledge for Dialogue Generation and Interpretation **(DRAFT)**

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 a new methodology for automatically compiling discourse knowledge during discourse planning. The resulting discourse knowledge can be used both in dialogue generation and interpretation. The methodology is based upon the learning mechanism of the Soar architecture.

Introduction

Discourse plan operators or recipes have been used successfully in computational systems for generating and interpreting discourse, e.g.,(Caw92; GC94; Hov88; LC91; LA87; May92; MP93). A considerable advantage of using discourse recipes is on-line efficiency, i.e., some non-trivial amount of problem-solving has been "compiled away" off-line. One purported disadvantage is lack of completeness. In other words, every case which could be handled by problem-solving may not be represented in a system's set of recipes. Thus, it might be claimed, a recipe-based system lacks the flexibility needed to model human dialogue. Furthermore, it might be claimed that, as more discourse recipes are added to a system, its performance will necessarily suffer.

In this paper, we provide a methodology for automatically compiling discourse knowledge during discourse planning. This compiled knowledge (which can be thought of as serving the same role as traditional discourse recipes) is used to speed up discourse generation without sacrificing flexibility and completeness. Note that whenever the store of compiled discourse knowledge fails to cover the current situation, our system falls back upon discourse planning, automatically acquiring a new "discourse recipe" in the process. Performance is not sacrificed as the store of compiled knowledge grows, due to the underlying architecture's support for this type of learning. Moreover, we demonstrate how this same compiled discourse knowledge can be exploited during discourse comprehension to enable the system to recognize the speaker's discourse intentions.

This approach has been implemented as part of the dialogue processing component of NL-Soar,[1] a computational model of natural language interpretation and generation implemented in Soar (New90; LLR96). Soar is an integrated problem-solving and learning architecture, and our methodology for acquiring compiled discourse knowledge is based upon Soar's basic learning mechanism. The current implementation of NL-Soar is being developed to provide real-time dialogue capabilities to automated intelligent agents who may converse with humans or other intelligent agents in air-combat simulation environments (TJJ$^+$95). Real-time behavior is achieved by training the system on representative dialogues, so that post-training, dialogues are generated using mostly compiled knowledge. (The training dialogues need not be identical to the actual dialogues encountered in real-time. Also note that compiled syntactic and semantic knowledge also contributes to NL-Soar's real-time behavior (GL96).)

The paper is organized as follows. Section 2 describes a sample dialogue. Section 3 describes the discourse planning component. Sections 4-6 describe the learning methodology and its use in generation and interpretation, respectively. Section 7 describes related work, and Section 8 offers our conclusions.

Sample Dialogue

Figure 1 shows a constructed dialogue which is representative of transcripts of naturally occurring dialogues between human participants in air-combat training simulations. Participants flew in a simulated plane and communicated with the other participants over a simulated radio channel. In the sample dialogue, there are two participants labeled P1 and P2, whose radio call-signs are Parrot101 and Parrot102, respectively. Note that as in other domains involving two-way radio discourse (Gib85), a large portion of each turn is devoted to discourse moves for regulating the turn

[1] NL-Soar has been developed over the last several years by a number of researchers. Earlier versions are described in (LLN91; LVDR95; Lew93; RL94). Currently, inputs and outputs to NL-Soar are written. Interfacing NL-Soar to a speech recognizer is an area of current research.

	UTTERANCE	DISCOURSE MOVE
P2:	Parrot101.	summons
	This is Parrot102.	self-introduction
	I have a contact bearing 260.	inform
	Over.	end-turn
P1:	Parrot102.	summons
	This is Parrot101.	self-introduction
	Roger.	acknowledge
	I have a contact bearing 270.	inform
	Over.	end-turn
P2:	Parrot101.	summons
	This is Parrot102.	self-introduction
	Roger.	acknowledge
	That is your bogey.	inform
	Over.	end-turn

Figure 1: Sample Dialogue

(SSJ74) and grounding (CS89). That is, the function of the summons and self-introduction is to identify the addressee and speaker, respectively, the function of the acknowledgment is to ground the preceding turn, and the function of the end-turn move is to explicitly mark the release of the turn. These moves are in service of the main communicative goal underlying the segment, which is to establish the mutual belief between P1 and P2 that P1 has identified the same entity that P2 described in the first turn.

Discourse Planning

The input to discourse planning[2] is the agent's current mental state, including (1) the *conversational record*, and (2) the agent's *private beliefs and desires*. The conversational record consists of information which an agent A1 presumes to be shared with another agent A2 as a result of being engaged in a sequence of dialogue exchanges with A2 (Tho90). It includes information such as the current discourse segment purpose (DSP) (GS86), turn management information, discourse expectations (Car90; Rei85), and the status of proposals which each agent has made with respect to the current DSP. An agent A1's private beliefs include information which A1 does not (yet) presume to be shared with another agent A2, e.g., information which A1 may use to satisfy a request from A2. Note that an agent's private beliefs are annotated with symbolic confidence-levels. An agent's private desire is a DSP which has not yet been entered into the conversational record, i.e., is not yet a shared goal. The output of discourse planning may include (1) a sequence of one or more discourse moves sent to a buffer for realization by a sentence generation component, and (2) modifications of the conversational record expected to result from the other agent's comprehension of the planned moves.[3]

Next, we briefly describe the discourse planning algorithm implemented in the current version of the system by approximately 35 *if-then* rules. Note that a variety of dialogues representative of the current application domain have been generated by this algorithm.[4] The algorithm plans one turn at a time.[5] A turn is planned in three stages. First, turn-taking and grounding moves are selected. For example, if speaking over a radio channel, then it is assumed that it is necessary to explicitly identify who is talking.

Second, illocutionary moves are planned based upon the contents of the conversational record and, possibly, a small number of types of private beliefs and/or a private desire. Aspects of the conversational record relevant to planning illocutionary moves include the current DSP (if known) and the current set of discourse expectations. Three DSPs are covered, namely: to achieve the mutual belief between agents A1 and A2 that (DSP1) A1 and A2 have identified the same entity E, (DSP2) A1 and A2 know the answer R to a *wh*-question Q (i.e., A1 addresses Q to A2, A2 provides R to A1, and A1 accepts R), and (DSP3) the addressee believes a proposition P.

Discourse expectations are evoked by illocutionary moves.[6] For example, the discourse expectations evoked by the *inform* in turn 1 of the sample dialogue in Figure 1 are: (E1) that the addressee will inform the speaker of a description of an object of the type in question, (E2) that the addressee will inform the speaker that the addressee cannot inform him of a description of an object of the type in question, or (E3) that the addressee will inform the speaker that the ob-

[2] Note that the inputs and outputs of discourse generation are the same regardless of whether achieved by discourse planning or by use of compiled discourse knowledge.

[3] If the other agent fails to comprehend the moves as intended, then detection of and recovery from the miscomprehension are required. However, that problem is beyond the scope of our research.

[4] The goals of this implementation are to provide coverage in the current application domain and to enable us to test the discourse knowledge compilation methodology. We believe that a more sophisticated planning algorithm such as described in (YMP94) could be substituted without substantially altering our approach of compiling discourse knowledge for use in generation and comprehension. For example, a hierarchical-task-network planner which has been implemented in Soar (Gra96) could be used in place of the planning operators used in the current implementation.

[5] Planning is performed one turn at a time, rather than over multiple turns, to provide responsiveness to the changing situational and discourse context. Note that to allow a greater responsiveness as well as to allow a finer-grained interleaving of planning and sentence generation, planning could be performed in even shorter increments.

[6] I.e., during comprehension, the conversational record's current set of discourse expectations is updated as each discourse move is recognized.

ject O1 that the addressee last described is identical to the object O2 of the same type that the addressee has in mind.[7] Seven types of illocutionary moves are planned in the current implementation.

To give an example, the *inform* move in the third turn of the sample dialogue would be planned by a rule saying that if the DSP is the one listed in (DSP1), and if the current set of discourse expectations includes the one listed in (E3), and if the agent has a private belief (with confidence level of certainty) that objects O1 and O2 are identical, then a move informing the addressee that O2 is the same as O1 is planned.

Lastly, turn-management moves are planned again (e.g., producing an explicit end-turn move).

Compiling Discourse Knowledge

Building D-plan-constructors

We refer to a compiled unit of knowledge about discourse in NL-Soar as a *d-plan-constructor*, which is produced as a side-effect of discourse planning.[8] This section describes NL-Soar's technique for learning d-plan-constructors, which makes use of Soar's built-in learning mechanism.[9] In order to describe the technique, it is necessary to first give a very brief overview of the relationship of problem-solving to learning in Soar.

In Soar, a task is performed by selecting and applying one or more *operators* in sequence. An operator is defined by a set of productions (*if-then* rules) describing conditions for its selection and conditional actions it may perform. Whenever no operator applies in the current situation, Soar signals that an *impasse* has occurred, and creates a *subgoal*. Within the subgoal, Soar's basic control structure continues to operate, which may give rise to a stack of subgoals. Whenever Soar detects a situation in which problem-solving in a subgoal results in changes to state in a supergoal, Soar automatically creates a *chunk*, a learned production. The left-hand (*if*) side of a chunk encodes what attributes of the supergoal's state were tested that led to the result, and the right-hand (*then*) side encodes the result. Then, if a similar problem is encountered again, Soar is able to use the chunk instead of repeating the original problem-solving.

Note that, in theory, a system designer could produce the same set of productions that are produced by chunking since chunks are encoded in the same representational format as hand-coded productions. However, in practice, that would make the system designer's task much more complex. Also, a system with both chunking and deeper problem-solving has the capacity to robustly handle cases not anticipated during the original design. Another advantage to this approach is that solving a problem via chunks may be much more efficient because the chunks enable the same results to be achieved with maximum parallelism. That is, all chunks whose left-hand sides match the current situation fire in parallel. Thus, data-dependencies are the only potential source of non-parallelism. Furthermore, due to Soar's underlying match algorithm, performance in most Soar systems (including NL-Soar) does not degrade as the number of learned productions grows (the so-called *utility problem* in machine learning) (Doo95).

In NL-Soar, the chunking process is used to create each d-plan-constructor, a learned operator roughly corresponding to a discourse recipe. The process of acquiring d-plan-constructors goes as follows. NL-Soar includes a top-level discourse planning operator, **learn-discourse**, which is designed to allow an impasse to arise. That is, a set of productions specify when it should be selected,[10] but do not specify how it should plan the turn. To illustrate the following discussion, an excerpt of an execution trace of NL-Soar from the point at which **learn-discourse** has been selected to the point at which a set of chunks making up a new d-plan-constructor (**d-plan-constructor1**) are executed is shown in Figure 2. In the trace, which has been edited for readability, indentation indicates subgoaling and **S:** precedes states and **O:** precedes operators. Comments are annotated with asterisks and lines have been numbered for reference in the following discussion.

When the impasse on the **learn-discourse** operator occurs (2), NL-Soar automatically creates a subgoal (3). The system includes productions which, on detecting the creation of this subgoal, (a) name the subgoal *create-d-plan*, (b) make a copy of the agent's conversational record and relevant private beliefs/desires in the subgoal, and (c) cause a newly created operator of type d-plan-constructor to be selected in *create-d-plan*. The newly created operator is given a unique name, e.g., **d-plan-constructor1** (4). Since no productions specify what a newly created d-plan-constructor should do when it is selected for the first time, another impasse arises (5). Another production names this subgoal *d-plan*, which triggers the discourse planning algorithm described in the previous section. The three phases of planning are implemented by operators named **acquire-turn** (6), **achieve-d-goal** (7),

[7] Discourse expectations are used for guidance but dialogues are not required to follow expectations. Also, when a speaker satisfies any member of the set of current expectations, the entire set is discharged.

[8] Note that we do not claim this to be a model of the acquisition of discourse knowledge by children. Also, the compilation of discourse knowledge during comprehension, whether for use in generation or comprehension, is beyond the scope of this work.

[9] The general technique is similar to that developed for learning other recognitional knowledge of language described in (Lew93; LVDR95).

[10] More specifically, whenever no d-plan-constructor is applicable and the agent is free to take the turn and either the agent has a private desire to communicate or there is a discourse expectation for the agent to communicate.

```
1 S: Top Language State
2 O: learn-discourse
3    ==>S: Create-d-plan * copy Top
                         structures *
4       O: d-plan-constructor1
5       ==>S: D-Plan * make changes
                     to copy: *
6          O: acquire-turn
7          O: achieve-d-goal
8          O: release-turn
9          O: return-results
10         Building chunk-1 through chunk-10
11      O: return-d-plan
12      Building chunk-11
13 Firing chunk-11 * propose
                  d-plan-constructor1 *
14 O: d-plan-constructor1
15 Firing chunk-1 through chunk-10
   * make changes in Top Language State *
```

Figure 2: Trace of Learning/Executing a D-plan-constructor

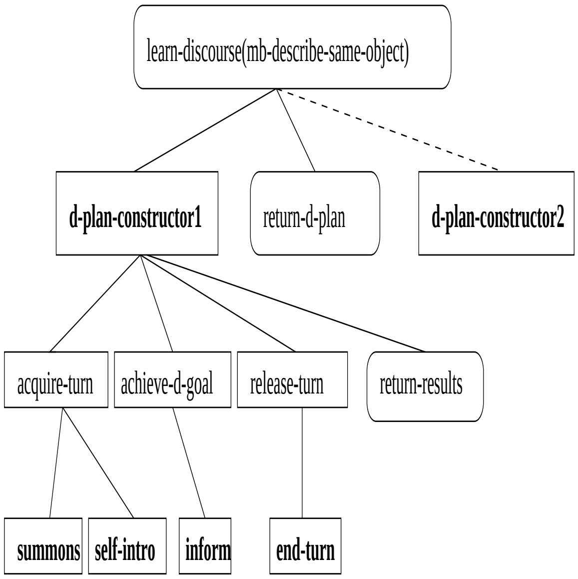

Figure 3: Hierarchy of Actions in Planning/Learning Turn 1

and **release-turn** (8), respectively.

At the end of planning, the **return-results** operator is selected in *d-plan* to return these results to *create-d-plan*, updating the copied agent state, e.g., specifying any planned discourse moves and changes to the conversational record (9). As a byproduct of returning results to the *create-d-plan* supergoal, NL-Soar creates a set of chunks that define the conditional actions of the new d-plan-constructor (10). Lastly, the **return-d-plan** operator in the *create-d-plan* subgoal returns the new d-plan-constructor to the Top state (11). A byproduct of this result is the creation of a chunk (12) that proposes the new operator for execution in the Top state (13). (This chunk also will enable the d-plan-constructor to be proposed in similar situations in the future, thereby preventing **learn-discourse** from being selected.) When the returned d-plan-constructor is run in the Top state (14), the chunks which implement the new d-plan-constructor, chunk-1 through chunk-10, match and thus make changes to the Top state discourse structures (15).

In summary, Figure 3 shows the hierarchy of actions involved in producing the first turn of Figure 1. Non-terminal nodes enclosed in rectangles represent discourse planning operators, while non-terminal nodes enclosed in ovals represent plan-compilation operators. Leaf nodes are abstract discourse moves to be realized by sentence generation. After plan-compilation is finished, only the nodes labelled in boldface remain. **D-plan-constructor2** is shown in the Figure connected by a dotted line to the root node to indicate that future discourse planning for this speaker will use this same mechanism during his next turn, working towards achieving the current DSP. In other words, the hierarchy of planning actions eventually dominated by this DSP can be thought of as this speaker's discourse plan.

D-plan-constructors

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 by chunking during discourse planning, rather than provided by the system builder.
- Whereas traditional discourse recipes are declarative specifications which must be interpreted at run-time, d-plan-constructors represent procedural knowledge which may be applied with a high degree of parallelism.
- 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 subgoal relations are not retained automatically in chunks by NL-Soar, a Soar program can be designed so that this type of information gets encoded in chunks. However, information about hierarchical and causal relations between actions is not used in the current implementation of the planner.
- Most traditional approaches to discourse processing (one exception is (GC94)) have not used the same discourse recipes for both generation and interpre-

tation. We describe in a later section how d-plan-constructors can be used in discourse interpretation.

To give a detailed example, we describe **d-plan-constructor1**, created during the planning of turn 1 as shown in Figure 2. This d-plan-constructor, like all learned operators in NL-Soar, is defined by a set of learned productions (chunks). A chunk telling NL-Soar when a d-plan-constructor is applicable to a state is referred to as a *proposal* chunk, and is roughly comparable to the applicability conditions (Car90) of a discourse recipe. Other productions, the *implementation* chunks, actually perform the state changes. We can paraphrase the condition (left-hand) side of chunk-11, which is the proposal chunk for **d-plan-constructor1**, as:

- state s1 allows language operators,
- the agent's private beliefs include a belief that the agent is certain that he can describe an object of type o1,
- according to the conversational record, there is currently no DSP, the medium is two-way radio, the domain is air-combat training simulation, and the turn is available,
- no moves are waiting to be realized, and
- the agent has a private desire that speaker and hearer mutually believe they are describing the same object, which is of type o1.

The action (right-hand) side of the chunk tells NL-Soar that **d-plan-constructor1** may be applied in s1.[11]

Once **d-plan-constructor1** has been selected by NL-Soar, its conditional actions may be performed. Note that in this case, there are no data dependencies and the chunks will fire in parallel. To describe the other chunks of this d-plan-constructor,

- chunk-1: records a DSP of *mb-describe-same-object* in the conversational record,
- chunk-2: creates the first move (*summons*) and adds it to the planned move buffer,
- chunk-3: does the same for the second move (*self-introduction*),
- chunk-4: does the same for the third move (*inform*),
- chunk-5: does the same for the fourth move (*end-turn*),
- chunk-6: records the proposal to be made by the third move in the conversational record,
- chunk-9: records information about the turn in the conversational record, and
- remaining chunks: set control flags.

To paraphrase chunk-2, for example, its left-hand side conditions are:

- operator o1 named **d-plan-constructor1** has been selected, and
- the conversational-record contains the information that the medium is two-way radio.

The action side of chunk-2 adds a planned move to the planned move buffer. The planned move is a summons, which is to be realized by saying the addressee's name. Also, chunk-2 specifies that the move is the first move of the turn, although sentence generation may override ordering specifications.

Dialogue generation

Whenever a d-plan-constructor is selected and applied in the Top Language State, generation is performed. (When no d-plan-constructor is applicable, i.e., if an appropriate d-plan-constructor has not yet been acquired via training, NL-Soar will invoke **learn-discourse** to create the d-plan-constructor and then apply it to the current situation, as described in the previous section.) For example, whenever a situation similar to the one in which **d-plan-constructor1** was learned arises,[12] it will be selected. The chunks of a d-plan-constructor add planned discourse moves to a buffer which is the input to sentence generation. The chunks also update the speaker's version of the conversational record appropriately. The process of dialogue generation via chunks is more efficient than dialogue generation via the planning algorithm due to the fact that the same results are achieved without performing the original problem-solving that led to the creation of the chunks (as described in the previous section). Moreover, the actions are performed with maximum parallelism. That is, as soon as a d-plan-constructor has been selected, all of its chunks whose left-hand sides match the current situation fire in parallel. Thus, data-dependencies are the only potential source of non-parallelism.

Dialogue interpretation

Here we describe an experimental implementation of discourse comprehension in NL-Soar which reuses the d-plan-constructors created for and used in discourse generation. Most traditional systems (even those using hand-coded recipes) have not used the same discourse recipes for generation and comprehension. However, the ability to reuse this knowledge in service of comprehension reduces problems in maintaining consistency of discourse knowledge between comprehension and generation subcomponents, and reduces the

[11] Note that, in Soar, all productions whose left-hand side is matched by the current situation will fire in parallel. Thus, it is possible for multiple operators to be proposed simultaneously. In NL-Soar, *search-control* productions are used to specify preferences among currently eligible operators in case of multiple proposals.

[12] More specifically, whenever the proposal conditions of chunk-11 are satisfied.

system builder's effort in providing knowledge for discourse comprehension.[13] Thus, it is an intriguing area of research of practical value.

The task of dialogue interpretation in NL-Soar is to update the (hearer's version of the) conversational record as intended by the speaker, based upon the conversational record and the semantic interpretation of the speaker's utterances. For example, given the semantic interpretation of an utterance U to be a name referring to the hearer (e.g., *Parrot102*, the first utterance of the second turn in Figure 1), then in an appropriate discourse context the hearer's task is to recognize that U is being used as a summons (as opposed to, e.g., a response to a *wh*-question, or an elliptical self-introduction), and that the intended effect is to update the conversational record with the mutual belief that the speaker (P1) has begun a new turn of which the hearer (P2) is the addressee. Note that if the hearer knew which of the speaker's d-plan-constructors was used to produce an utterance, the hearer could update his version of the conversational record as specified by that d-plan-constructor.

In the experimental version of NL-Soar, a hearer's d-plan-constructors are used as a knowledge source for dialogue interpretation. This presupposes that the hearer could generate anything that he could interpret, i.e., that he has a set of d-plan-constructors comparable to the speaker's.[14] Such an assumption is consistent with the view of discourse interpretation as intended plan recognition (PA80). Moreover, this assumption is reasonable in practice given that NL-Soar's approach to providing real-time generation is to acquire a set of d-plan-constructors through prior off-line training. To give an overview of our strategy for discourse comprehension using compiled knowledge, NL-Soar (acting as the hearer) constructs an initial model of the speaker's relevant beliefs and desires, which we refer to as the Hearer's Model of the Speaker (HMOS). Then, in the context of HMOS, d-plan-constructors are automatically selected and applied by NL-Soar, resulting in updates to HMOS. In other words, the speaker's process of discourse generation is simulated in HMOS. NL-Soar monitors changes to HMOS and makes changes to the (hearer's version) of the conversational record accordingly.

Note that we have not yet addressed strategies for handling detection of and recovery from failure, i.e., inferring the wrong d-plan-constructor.[15] Thus, when more than one d-plan-constructor is recognized by NL-Soar to be applicable to the current state of HMOS, interaction with the system designer is used to select the right d-plan-constructor. This has enabled us to verify that, given the correct choice of d-plan-constructor, the proposed mechanism is successful in recognizing the speaker's intentions in dialogues such as that shown in Figure 1 and others.

Hearer's model of the speaker

The HMOS consists of the hearer's model of the speaker's version of the conversational record and the hearer's model of the speaker's relevant private beliefs and desires. Ideally, the HMOS must contain sufficient information to trigger the d-plan-constructor used by the speaker, but not information which would lead to recognition errors. (An alternate approach to elaborating the HMOS is to eliminate conditions on the left-hand side of the d-plan-constructor chunks, using a technique such as given in (Mil93).) In our initial implementation, we are investigating the utility of some minimal heuristics.

First, since the conversational record is presumed by the hearer to normally consist of shared information, the HMOS conversational record is created by simply copying the hearer's version of the conversational record. Second, the HMOS private beliefs is constructed using the following heuristics.

- The roles of speaker and hearer are reversed.
- The speaker is assumed to have a belief corresponding to the literal semantic interpretation of the utterance under consideration.
- If not currently engaged in a discourse segment (according to the conversational record), then a possible discourse goal type is selected at random.
- A set of salient beliefs is assumed based upon the current discourse segment purpose, if available from the conversational record; otherwise, based upon the the assumed discourse goal type. Note that salience is defined operationally relative to the set of d-plan-constructors which are applicable to the current assumed discourse goal type.[16]

We certainly recognize that, as heuristics, the above may not always provide an accurate model of the speaker. For example, obviously the assumption that the literal semantic interpretation is the intended interpretation is not always correct. Investigation of refinements of the current heuristics is an area for future research.

[13] The non-experimental version of discourse comprehension, i.e. the version currently in use in NL-Soar, consists of hand-coded productions which duplicate much of the same knowledge specified in the discourse planning component.

[14] That is, this presupposes that hearer and speaker share knowledge about discourse strategies, though not necessarily other knowledge about the world.

[15] However, we envision a general strategy consistent with that used in error recovery in sentence comprehension in NL-Soar, which is intended to model real-time human performance limitations.

[16] Currently, the association between each goal type and set of salient beliefs is hand-coded. We intend to implement a mechanism so that this association is learned at the same time that the d-plan-constructors are learned.

Discourse Interpretation Algorithm

The high-level discourse interpretation operator, **comp-d-move**, is selected whenever an utterance has been semantically and syntactically interpreted by NL-Soar. Note that during the process of syntactic and semantic interpretation, features are assigned to an utterance describing the *surface speech act*[17] (AP80) and the general *content type*, an abstraction of the semantic structure. First, **comp-d-move** checks whether the current utterance has been predicted by a d-plan-constructor recognized previously. (For example, if a d-plan-constructor which generates four moves is recognized from the first move, then **comp-d-move** need check only whether the succeeding three utterances match the remaining predicted moves.) If not, the operator constructs the HMOS as described in the preceding section. For illustration, consider the interpretation of the first utterance of turn 1 of the sample dialogue. As soon as the HMOS has been constructed, any d-plan-constructors whose applicability conditions are satisfied by the HMOS will be proposed by NL-Soar. As discussed above, interaction with the system designer is required in the current implementation to select the appropriate d-plan-constructor from the set of eligible operators, in this case, **d-plan-constructor1**.

When a d-plan-constructor has been selected, its execution chunks fire, performing updates to the HMOS conversational record and adding the speaker's planned discourse moves to the sentence generation buffer in HMOS. (Note that more than one planned move may be generated by a d-plan-constructor.) Next, **comp-d-move** compares the content type and surface speech act of the utterance to those features of the predicted moves.[18] In this example, the utterance matches the features of a predicted summons and so is recognized. The other moves predicted by **d-plan-constructor1** are copied into the agent's top state (as predictions), his conversational record is updated to reflect the changes in the HMOS conversational record, and the **comp-d-move** operator terminates.

As soon as each of the remaining utterances of the turn have been semantically and syntactically interpreted, **comp-d-move** is selected again. As long as there are predicted discourse moves to consider, there is no need to attempt to recognize another d-plan-constructor. Instead, the current utterance is compared to the predicted moves in order to be recognized.

Related work

In addition to the related work on discourse cited throughout this paper, our approach to recognizing discourse intentions has been influenced by earlier work on event tracking implemented in Soar (TR95; Tam96). In that work, an agent's own operators and architectural mechanisms are used to simulate the behavior of another agent in order to comprehend the other agent's non-communicative actions. However, that work does not involve use of learned operators for generating or recognizing behavior. Another difference is that NL-Soar models intended plan recognition whereas the event tracking work does not. (For a more detailed comparison, see (GL96).)

[17] E.g., the surface speech act of an utterance in the form of a question is a request.

[18] Although the planning algorithm specifies a preferred ordering of moves, this information is not used to constrain interpretation currently.

Conclusions

We provide a new methodology for automatically compiling discourse knowledge during discourse planning. This compiled knowledge (which can be thought of as serving the same role as traditional discourse recipes) is used to speed up discourse generation without sacrificing flexibility and completeness. Compiled discourse knowledge enables discourse generation to be performed more efficiently because the results of planning are achieved without deep problem-solving and with maximum parallelism. Moreover, we demonstrate that the compiled discourse knowledge can be exploited during discourse comprehension to enable the system to recognize the speaker's discourse intentions. In addition to its practical benefits, the work raises many interesting issues in cognitive modeling of dialogue processing that we would like to address in the future.

Acknowledgments

We thank Martha Pollack of the University of Pittsburgh and Milind Tambe of Information Sciences Institute for feedback on a much earlier version of this paper. Also we thank Garrett Pelton and Greg Nelson of the CMU Soar community for their programming advice during the implementation of the system. 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.

References

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

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

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

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

Robert B. Doorenbos. *Production matching for large learning systems*. PhD thesis, Carnegie Mellon University, 1995.

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

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

Nancy Green and Jill Fain Lehman. Comparing agent modeling for language and action. In *Proceedings of the AAAI Workshop on Agent Modeling*, 1996.

Jonathan Gratch. Task-decomposition planning for command decision making. In *Proceedings of the 6th Conference on Computer Generated Forces and Behavorial Representation*, 1996.

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

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

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

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

Richard Lawrence Lewis. *An Architecturally-based Theory of Human Sentence Comprehension*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1993. Available from Computer Science Department as Technical Report CMU-CS-93-226.

J. Fain Lehman, R. Lewis, and A. Newell. Integrating knowledge sources in language comprehension. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 1991.

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

J. F. Lehman, J. Van Dyke, and R. Rubinoff. Natural language processing for ifors: Comprehension and generation in the air combat domain. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, pages 115–23. 1995.

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

Craig Miller. *Modeling Concept Acquisition in the Context of a Unified Theory of Cognition*. PhD thesis, University of Michigan, 1993. CSE-TR-157-93.

Johanna D. Moore and Cecile Paris. Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, 1993.

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

R. Perrault and J. Allen. A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, 6(3-4):167–182, 1980.

Rachel Reichman. *Getting Computers To Talk Like You And Me*. MIT Press, Cambridge, Massachusetts, 1985.

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

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

Milind Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.

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

Milind Tambe, W. Lewis Johnson, Randolph M. Jones, Frank Koss, John E. Laird, Paul S. Rosenbloom, and Karl Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1):15–39, Spring 1995.

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

R. Michael Young, Johanna D. Moore, and Martha E. Pollack. Towards a principled representation of discourse plans. In *Proceedings of the Sixteenth Annual Meeting of the Cognitive Science Society*, 1994.