A Computational Model of Collaboration on Referring Expressions

Peter Anthony Heeman

Technical Report CSRI-251
September 1991

Computer Systems Research Institute
University of Toronto
Toronto, Canada
M5S 1A1

The Computer Systems Research Institute (CSRI) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is an Institute within the Faculty of Applied Science and Engineering, and the Faculty of Arts and Science, at the University of Toronto, and is supported in part by the Natural Sciences and Engineering Research Council of Canada.

A Computational Model of Collaboration on Referring Expressions

by

Peter Anthony Heeman

Department of Computer Science
University of Toronto
Toronto, Canada
M5S 1A4
September 1991

A thesis submitted in conformity with the requirements
for the degree of Master of Science at the
University of Toronto

Abstract

In order to refer to an object, a speaker attempts to build a description of the object that will allow the hearer to identify it. Since the description might not enable the hearer to identify the referent, the speaker and hearer might engage in a clarification subdialogue in which they collaborate in order to make the referring action successful.

This thesis presents a computational model of how a conversational participant collaborates in order to make a referring action successful. The model is based on the view of language as goal-directed behaviour. As such we propose that the content of a referring expression can be accounted for by the planning paradigm. Not only does this approach allow the process of building referring expressions and identifying their referents to be captured by plan construction and plan inference, it also allows us to account for how participants clarify a referring expression by using meta-plans that reason about and manipulate the plan derivations corresponding to referring expressions. To complete the picture, we show how participants can infer goals underlying referring expression plans or clarification plans, and how these inferred goals can be used by the participants to adopt their own goals to clarify the referring expression. An important aspect of this process is that subsequent clarifications can either clarify the previous clarification or can clarify the referring expression resulting from the previous clarification. So for the latter case, the same meta-plans that are used to clarify the original referring expression can be used to model the subsequent clarifications.

Acknowledgments

I would like to thank my co-supervisors Graeme Hirst and Janyce Wiebe, who have provided me with guidance and support, both for my thesis and for my future career.

I would also like to thank Diane Horton and the rest of the students in the Natural Language Group at the University of Toronto. Their feedback greatly improved the quality of my thesis.

Thanks to all of my friends in the Department of Computer Science and elsewhere, especially Ray, Dimitri, Greg and Andre, Rita, and Rob. You listened to me talk about my thesis work, and some of you even pretented to be interested in it!

Thanks especially to my family, who have stood by me through the years.

Financial support was gratefully received from the Natural Sciences and Engineering Research Council of Canada.

Contents

List of Figures

Chapter 1

Introduction

One way of viewing people is that we are goal oriented and we can plan courses of actions to achieve our goals. But sometimes we might lack the knowledge needed to formulate a plan of action, or some of the actions that we plan might depend on coordinating our activity with other agents. How do we cope? One way is to work together, or *collaborate*, in formulating a plan of action with the people who will be doing the actions or who know the missing information.

In this thesis, we adopt the view that language is goal-oriented behaviour in which the utterances are actions that are being used to achieve a goal. The success of these actions relies on both the speaker and the hearer, since the speaker must construct an utterance in such a way that the hearer can understand it. If the speaker is not successful, she can collaborate with the hearer in formulating a plan of action to achieve her goal. In this case, the actions are utterances and are executed during the course of the collaboration.

Let's look at referring expressions. The speaker has the goal of wanting the hearer to identify an object that the speaker has in mind. The speaker attempts to achieve this goal by constructing a description of the object that she thinks will enable the hearer to identify it. But since the speaker and the hearer have different beliefs about the world, the hearer might not be able to identify the object. When the hearer cannot do so, the speaker and hearer can collaborate in making a referring expression that accomplishes the goal.

1.1 Our Work

This thesis presents a computational model of how a conversational participant collaborates in making a referring action successful. We use as our basis the model proposed by Clark and Wilkes-Gibbs (1986). In their work, they give a descriptive

account of the conversational moves that participants make when collaborating to make a referring expression. In making it computational, we draw on the insights of Appelt (1985a), Litman and Allen (1987), and Pollack (1986), and propose a model based on the planning paradigm. From Appelt, we use the idea that the act of referring can be incorporated into a plan-based model of how agents construct utterances. From Litman and Allen, we use the idea that *meta-plans* can account for how conversational participants clarify an utterance. And from Pollack, we use her idea that plans are mental objects, which facilitates the inference of invalid plans.

We propose that referring expressions can be represented by plan derivations, and that plan construction and plan inference can be used to generate and understand them. Not only does this approach allow the process of building referring expressions and identifying their referents to be captured in the planning paradigm, it also allows us to use the planning paradigm to account for how participants clarify a referring expression. In this case, the plans are meta-plans that reason about and manipulate a plan derivation corresponding to the referring expression.

To complete the picture, we account for how an agent, after it has inferred the plan underlying an utterance, adopts goals of its own to collaborate in making a referring action successful. An important aspect of this process is that subsequent clarifications can either clarify the previous clarification or can clarify the referring expression resulting from the previous clarification. So for the latter case, the same meta-plans that are used to clarify the original referring expression can be used to model the subsequent clarifications.

This thesis makes several contributions to the field. First, although much work has been done on how agents request clarifications and how they clarify, this is not the same as collaborating. This work is a start in formalizing what collaboration is, both in terms of the goals and intentions that underlie it and the surface speech acts that result from it. Second, it addresses the act of referring and shows how it can be better accounted for by the planning paradigm, thus contributing towards answering Allen's question about "where this type of reasoning fits into the planning framework" (1979, p. 117). Third, previous work has concentrated on either construction or recognition, but not both. By doing both, we have ensured that our work scales up in the direction of providing a complete model of the collaborative process. Finally, by using Clark and Wilkes-Gibbs's model as a basis of our work, we not only add support to their work and our own work, but gain a much richer understanding of the subject.

1.2 Scope

In order to address the problem that we have set out, we have limited the scope of our work. This has been done so that we can concentrate on the problem of how agents collaborate, without getting into a great number of other problems that would lead as astray of our research objective.

First, we look at referring expressions in isolation, rather than as part of a larger speech act. One advantage of doing this is that we do not need to recognize which goal it is that an agent is collaborating on, since it can only be the referring action.

Second, we assume that the plan library is shared. In other words, agents have mutual knowledge about referring expressions and about how agents collaborate in making them. Since these are communicative plans rather than domain plans, this assumption does not seem that limiting.

Third, we deal with objects that both the speaker and hearer know of. Objects are represented as constants such as `bird1` and an agent's beliefs about an object are represented by propositions such as `colour(bird1,brown)`. The act of identifying the referent is to match the initiator's description of the object with an object that the responder knows about. However, we do not assume that both the initiator and the responder have the same beliefs about the objects. This class of referring expressions roughly corresponds to Appelt's *shared concept activation with identification intention* (1985c).

Fourth, as the input and the output to our system, we use surface speech actions. We assume the existence of other modules that can parse an utterance into these actions and generate utterances from them.

Fifth, we are not attempting to handle all types of utterances that might occur when collaborating to make a referring action successful. We are attempting to give a computational model of the collaborative process, rather than providing wide coverage.

Finally, we do not specifically address the issue of how participants' beliefs are changed during the acceptance of a referring expression. To properly account for this, we would need to employ a model of belief revision and also account for an agent's confidence in its beliefs. Given this restriction, the collaborative moves that we account for are such that they only need to augment beliefs rather than revise them.

1.3 Some Terminology

Throughout this thesis, we need to refer to the conversational participants or agents involved in the discourse. Although we could use those terms, it is more beneficial to use terms that capture the role that each agent is performing in the discourse. So, we use the term *initiator* to refer to the agent who initiates the referring action, and the term *responder* to refer to the agent who is trying to identify the referent. Also, it will be helpful to use terms that are relative to an utterance. For this we use the terms *speaker* and *hearer*, where *speaker* is used to refer to an agent who is adopting goals to be achieved in an utterance or who is constructing an utterance, and *hearer* refers to an agent who is inferring the plan and goal that underlie an utterance or who is adopting beliefs based on the utterance. When illustrating our system, we use the terms *system* and *user*.

Since we will need pronouns to refer to the agents, we will adopt the following convention. Initiator and speaker will be referred to using the pronoun *she*, and responder and hearer will be referred to using the pronoun *he*.[1]

We also need to present our planning terminology.[2] We will be using the terms *plan schemas*, *plan instances*, *plan derivations*, *plan construction*, and *plan inference*. A plan schema consists of a *header*, *constraints*, a *decomposition*, and an *effect*;[3] and it encodes the constraints under which an effect can be achieved by the plan headers and primitive actions in the decomposition. A plan instance is an instance of a plan schema in which some of the variables in the plan might be instantiated. A plan derivation is a plan instance in which the plan headers in its decomposition have been recursively expanded into plan instances. Plan construction is the process of finding a plan derivation in which the primitive actions of the plan derivation (*yield*) will achieve a given effect, and plan inference is the process of finding a plan derivation whose yield is a set of observed primitive actions. Finally, where the context allows, we will use the word *plan* to refer to either a plan schema, a plan instance, or a plan derivation.

1.4 Overview of the Thesis

In the next chapter, we discuss the relevant literature.

---

[1] When an agent switches from understanding the previous utterance (hearer) to constructing the next utterance (speaker), the sex of the agent changes as a side effect.

[2] See Allen (1987) for an introduction to using plan-based models in natural language understanding.

[3] We do not consider preconditions.

In chapter 3 and 4, we focus on the part of our model that is based on the planning paradigm. We show how plans that underlie an utterance are constructed and inferred in isolation to how these processes fit into our model of goal-directed behavior that accounts for the process of how a referring action is made successful. Chapter 3 shows that the initial referring expression can be built and understood in the planning paradigm. We present plan schemas and surface speech acts that the implemented system uses and give examples of the system constructing and inferring referring expression plans. Chapter 4 shows that clarifications of the initial referring expression can also be built and understood in the planning paradigm. In this case the plans are meta-plans that take as a parameter a plan derivation that corresponds to a referring expression. In addition to presenting plan schemas and surface speech actions, we present predicates for reasoning about and manipulating the referring expression plans. Finally, we give examples of the system constructing and inferring clarification plans.

The purpose of chapter 5 is twofold. First, we show that subsequent clarifications of a referring expression can be accounted for by the same meta-plans that account for clarifications of the initial referring expression. Second, we show how the construction and inference of referring expression plans and clarification plans fit into a model of goal-directed behaviour. The goals for clarifying a referring expression arise from the mutual responsibility that the participants have towards making the referring action successful. The mutual responsibility also provides a context for understanding clarifications, and sanctions the updating of the common ground.

In chapter 6, we present a detailed example of the processing of our system on an entire dialogue. Chapter 7 discusses the implementation. In particular, we show how we have used the technique of chart parsing for plan inference, and how the same plan schemas can be used for both plan construction and plan inference.

In chapter 8, we summarize our contributions, discuss how it relates to previous work that has been done, and describe possible extensions.

Chapter 2

Foundations

This chapter reviews the foundations that our thesis is built on. Since our work proposes a model based on the planning paradigm to account for how people collaborate in making a referring action successful, the foundational work naturally divides itself into a number of areas, and each is reviewed in a section of this chapter. The first section reviews work that has been done in psychology on how people collaborate. The second section reviews computational approaches to generating and understanding the content of referring expressions. The third section reviews the planning literature relevant to representing what a conversation is about, specifically where agents might have invalid beliefs or incomplete beliefs with respect to constructing a plan. Finally, the fourth section reviews the planning literature relevant to communicative actions.

2.1 Psychological Investigations of Collaboration

As mentioned in the first chapter, our work recasts into a computational model the work done by Clark and Wilkes-Gibbs (1986) on how people collaborate in building referring expressions. In this section, we discuss their work along with two other related pieces of research.

2.1.1 Clark and Wilkes-Gibbs

Clark and Wilkes-Gibbs (1986) investigated how conversation participants collaborate in making a referring action successful. They conducted experiments in which participants had to refer to objects—tangram patterns—that are difficult to describe. They found that typically the initiator would first present an initial referring expression. The responder would then pass judgement on it, either accepting it, rejecting it, or postponing his decision until it was expanded. If it was rejected or the decision

postponed, then either the initiator or the responder would refashion the referring expression. This would take the form of either repairing the expression by correcting speech errors, expanding it by adding further qualifications, or replacing the original expression with a new expression. This process would be repeated until the initiator and responder mutually accepted the referring expression.

Below are two excerpts from their work that illustrate the acceptance process.

(2.1) **A:** [1] Um, third one is the guy reading with, holding his book to the left.

B: [2] Okay, kind of standing up?

A: [3] Yeah.

B: [4] Okay.

In the above dialogue, person A makes an initial presentation in (1). Person B postpones his decision in (2) by voicing a "tentative okay", and then proceeds to refashion the referring expression. A accepts the new referring expression in (3), and B signals his acceptance in (4).

(2.2) **A:** [5] Okay, and the next one is the person that looks like they're carrying something and it's sticking out to the left. It looks like a hat that's upside down.

B: [6] The guy that's pointing to the left again?

A: [7] Yeah, pointing to the left, that's it! (laughs)

B: [8] Okay.

In the second dialogue, B implicitly rejects A's initial presentation by replacing it with a new referring expression in (6). A then accepts the refashioned referring expression in (7).

Clark and Wilkes-Gibbs claim that by collaborating in the above fashion, participants can overcome a number of limitations that they are faced with.

- **Ignorance:** The initiator might not know enough about the responder to be able to determine if the responder will be able to uniquely identify the object.
- **Time Pressure:** Conversation must flow at a certain rate in order to be nondisruptive. The initiator might not be able to construct an appropriate noun phrase within this time limit.

- **Complexity:** The noun phrase that the initiator constructed might be too complex to be easily understood.

Clark and Wilkes-Gibbs also propose that conversational participants try to minimize *collaborative effort*. There is a trade-off between initiating a referring expression and refashioning it. The initiator might realize that it would take more collaborative effort to construct a proper referring expression than the effort involved in refashioning it with the help of the responder.

Clark and Wilkes-Gibbs argue that, in carrying out these clarification subdialogues, the participants establish a common perspective for describing the objects that they are referring to, and that the common perspective helps to minimize the collaborative effort. During the first round of presentations, they found that the initiator would try to establish a common perspective. The perspective would be refined during subsequent references. This resulted in a decrease in the amount of collaboration, as the common perspective between the participants facilitated future referring expressions.

2.1.2 Garrod and Anderson

Garrod and Anderson (1987) conducted experiments in which participants had to describe their location in a maze game. Garrod and Anderson analyzed the participants' descriptions over the course of several games. They found that participants adopted a common perspective for describing their positions. They claim that by adopting this common perspective, participants are trying to minimize the collaborative effort involved in describing their positions. To minimize the effort, it is necessary to be locally consistent (consistent from one turn to the next) in the way positions are described. This in turn causes the participants to adopt the same mental model of the maze and the same local semantics for descriptions, and so adopt a common perspective.

2.1.3 Clark and Brennan

Clark and Brennan (1990), expanding on the work of Clark and Wilkes-Gibbs, addressed the issue of *grounding* in conversation. Grounding is the process whereby participants try to establish that they have understood what has been said, so that their common ground can be updated. Participants do this by giving positive evidence that they have understood each contribution to the conversation. This can take the form of explicit acknowledgement such as "uh huh" and "yeah," a relevant next turn, for instance giving an appropriate answer to a question, or continued attention.

2.2 Referring Expressions

This section reviews the computational methods that have been used for generating and understanding the propositional content of referring expressions. All of the methods assume a knowledge base containing facts about the objects in the world.

2.2.1 Generating Referring Expressions

As part of a system that uses planning techniques to generate sentences, Appelt (1985a) addresses the issue of generating referring expressions. The content of a referring expression is chosen by an algorithm that corresponds to a primitive action in his plan library. The algorithm first chooses a basic category descriptor. It then adds descriptors based on facts about the object that are linguistically realizable and that are mutually known by the speaker and hearer. Descriptors are added until the object is uniquely identified according to the mutual knowledge of the two participants. Appelt also incorporates a simple model of context. If the object being referred to is the discourse center then the object does not have to be distinguished from the other objects.

Dale (1989) also uses an algorithm to construct referring expressions. His method is similar to Appelt's, but he chooses the descriptors in order of their discriminatory power. The descriptor that rules out the most candidates is added to the description until the description uniquely identifies the object being described.

Ehud Reiter (1990) also addresses the issue of determining the content of referring expressions. His work focuses on avoiding conversational implicatures in descriptions in a tractable manner. He does this by formulating three preference rules: the description should not have any unnecessary components, there should be no set of components that can be replaced by a single component, and that lexically-preferred classes should be used. For each of these rules, he presents a polynomial algorithm that will take a set of descriptors and choose a subset that meets the preference rule.

Reiter's work points out that when referring expressions are constructed, care must be taken to avoid conversational implicature. However, his solution, as have Appelt's and Dale's, relies on special-purpose algorithms for choosing the content of referring expressions, as opposed to integrating this aspect into a general theory of how agents determine the content of an utterance. Also, all three algorithms make strong assumptions about the speaker's knowledge of the hearer. Appelt assumes that the speaker has complete knowledge of the hearer's beliefs while Dale and Reiter do not take belief into account and so assume that the speaker and hearer have the same

beliefs. From the work done by Clark and Wilkes-Gibbs (1986), these assumptions seem unrealistic.

2.2.2 Understanding Referring Expressions

An obvious approach to understanding referring expressions is to regard this task as a constraint satisfaction problem. This technique is employed by Mellish (1985) in his system that understands physics problems. In his system, when a referring expression is encountered, a set of candidate objects is created. Each component of the description supplies a constraint; plus, constraints are derived from other places in the text such as co-referring descriptions. As these constraints are found, they are applied to the candidate set to derive the object that is being referred to. By the end of the text, the referring expression should only have a single candidate that matches the description.

The constraint satisfaction approach assumes that as a result of applying the algorithm, there will be only one object that matches the referring expression. Although this might work for understanding textbooks, in which assumptions can be made about the adequacy of referring expressions, this approach will fail in situations where the speaker and hearer do not share the same beliefs. The approach might find no objects, or multiple objects.

Goodman (1985) addresses the situation in which no objects match the description. He views descriptions as being approximate, and his computational model relaxes the literal content of the description in order to find a referent that will match it. The first step of his algorithm returns a set of candidate referents that are close to the description in terms of a knowledge base taxonomy. The second step orders the set of candidates in terms of how likely the candidates are to be the referent. This ordering reflects heuristic knowledge drawn from a number of knowledge sources, including linguistic knowledge, hierarchical knowledge, and perceptual knowledge. For instance, the linguistic knowledge base prefers relaxing adjectives, then prepositional phrases, then relative clauses and predicate complements. The candidates are ranked in terms of fewest number of components that need relaxing and how preferred those components are to relax. The third step attempts the actual relaxation, starting with the most preferred candidate. Again, multiple heuristics are used from multiple knowledge sources. For instance, if the candidate was turquoise, but the original description mentioned the colour blue, then the colour blue can be relaxed to turquoise on the basis of perceptual knowledge. Or, if the candidate is not blue but has a blue subpart, a heuristic based on hierarchical knowledge could relax the colour so that it only applied to the subpart. If the relaxation fails to relax the description to the

candidate, the next candidate is tried. This is repeated until a candidate is found.

Goodman's algorithm is an improvement over the constraint satisfaction approach, for it relaxes the assumptions that the speaker and hearer have exactly the same beliefs about the world. However, his algorithm assumes that the hearer has no recourse to question the speaker about the identify of the referent.

2.3 Planning

The planning literature that is relevant to this thesis has been separated into two sections. This section reviews the literature that is relevant to representing what a conversation is about, where agents might have invalid beliefs or incomplete beliefs with respect to constructing a plan. The next section reviews using planning techniques to account for communicative actions.

2.3.1 Invalid Plans and Plan Repair

Plans are sometimes invalid. An agent might have incorrect beliefs about the world, and hence it might construct a plan that will not achieve the goal that it is supposed to achieve. Plan repair is the process where either the planning agent or another agent fixes the plan so that it will achieve its goal.

Pollack (1986) has extended plan recognition to infer invalid plans that a user might have. She views plans as "mental objects" rather than as data structures. Under this view, "having a plan" means that an agent has a particular configuration of beliefs and intentions. In inferring a plan, she uses plan inference rules that capture the notion that the user might have incorrect beliefs. These inference rules include inferring a plan where the enablement conditions are not true, where the user might have mistakenly used an action that is similar to an action that the system believes will achieve the goal, used a plan that is similar to one that will achieve the goal, or made a typical mistake. To guide the plan inference process, both the goal that the user is trying to achieve and the domain action that the user thinks will achieve the goal are input to the plan inference system.

2.3.2 Shared Plans

Grosz, Sidner, and Lochbaum (Grosz and Sidner, 1990; Lochbaum, Grosz and Sidner, 1990) present a model of plans to account for how agents with partial knowledge collaborate in the construction of a domain plan. Agents have a library of partially specified plan schemas (recipes). These recipes might be underspecified as to how an

action is executed or how an action contributes to a goal. Agents then collaborate in constructing a shared plan by uttering statements about their beliefs and intentions about the plan. This collaboration will terminate with each agent mutually believing that each act in the plan can be executed by one of the agents, that that agent intends to perform the act, and that each act in the plan contributes to the goal.

Their computational work focuses on understanding how an agent's utterance contributes to the shared plan. They present an algorithm for augmenting the beliefs about the plan being collaborated on. If the inferring agent detects a belief about an action in the plan that is inconsistent with its own beliefs, it will communicate dissent rather than negotiating about the action.

2.4 Communicative Plans

This section reviews how planning techniques have been used to model communicative acts, with an emphasis on acts of reference and clarification subdialogues.

2.4.1 Speech Acts

By viewing language as action, the planning paradigm can be applied to natural language processing. The actions in this case are *illocutionary speech acts* (Grice, 1957; Austin, 1962; Searle, 1969), and include such things as promising, informing, and requesting. Cohen and Perrault (1979) developed a system that uses plan construction to map an agent's goals to speech acts, and Allen and Perrault (1980) use plan recognition to understand an agent's plan from its speech acts.

2.4.2 Referring as Action

Treating referring as a speech act was first proposed by Searle (1969). He proposed that referring and predicating were *propositional speech acts* as opposed to illocutionary acts. This distinction is due to Searle's view that referring and predicating cannot be done alone, but only as part of an illocutionary act.

Cohen (1981; 1984), differing from Searle, advocates treating referring as requesting. From analyzing task-oriented dialogues, he proposed the use of an action `identify(agt,D)`, which means that the hearer, `agt`, identifies the referent of the description, `D`. His approach would make identifying a referent the same as doing a physical action. A speaker would plan that the hearer identify a referent and request him to do so in the same way as the speaker would do for physical actions. So, models that account for the generation and understanding of requesting can be used for

referent identification.

Appelt and Kronfeld (1987) support Searle's view that referring is a propositional speech act and provide a formal model of referring. They propose that the literal goal of referring "is to establish mutual belief that a particular object is being talked about" and that the condition of satisfaction is for the hearer to "know the appropriate criteria for correct identification and successfully satisfy these criteria." The literal goal does not involve the hearer identifying the referent, but just having a representation of it. This is achieved by the hearer recognizing the intention of the referring goal. The criterion for identifying the referent depends on the illocutionary act among other things, and will take the descriptive content of the noun phrase into account.

2.4.3 Planning Referring Expressions

In a section 2.2.1, we discussed how Appelt (1985a; 1985b) chooses the content of referring expressions. In this section, we are interested in how acts of reference fit into his plan-based model of sentence generation.

In Appelt's model, after a surface speech act has been planned, it is passed to a functional unification grammar. The grammar is augmented so that when building a noun phrase for a term O_i of the surface speech act, it checks to see whether the term has a description (propositional content). If it doesn't then the grammar poses the goal `active`(O_i) to the planner. The proposition `active`(O_i) means that O_i "belongs to the available set of terms for constructing the proposition the speaker is conveying in way of the surface speech act" (1985a, p. 18). The planner achieves this goal by way of a *concept activation* action, which can be viewed as a generalization of a referring speech act. Concept activations are achieved by either the nonlinguistic action of pointing or by the linguistic action `describe`. The preconditions of `describe` specify that the speaker believes that the description is true of the intended referent, and that the intended referent is the only object that is not ruled out according to the mutual knowledge of the two agents. The `describe` action calls the algorithm that was mentioned in section 2.2.1 to choose a description that satisfies the preconditions. The chosen description is then used by the grammar to determine the linguistic realization of the noun phrase.

2.4.4 Discourse Plans and Clarification Subdialogues

Litman and Allen (1987) have taken a plan-based approach to understanding discourse. In addition to using domain plans to encode knowledge about the topic

of conversation, they use discourse plans to encode knowledge about conversations. Discourse plans correspond to how an utterance can relate to the current topic of conversation, be it a domain plan, or even another discourse plan. This approach allows them to use plan recognition to identify discourse phenomena from utterances, such as introducing a new topic, continuing a topic, and clarifications.

The plan recognition process infers the user's discourse plan from the utterance that it observes. It then tries to determine the plan that underlies the discourse plan. If this is not the first utterance, the plan recognition process will have as input a plan stack that represents the current context. The plan recognition process prefers interpretations that relate the discourse plan to one already on the stack. After the plan recognition process is complete, a new plan stack will be built and this is used to both determine the system's next action and provide context for the next statement.

Litman and Allen present three clarification plans that can account for a number of clarification subdialogues. To support these plans, they provide a vocabulary for referring to and manipulating the underlying plans. An example is `correct-plan` shown in figure 2.1. This plan takes another plan as a parameter (the underlying

Header:	`CORRECT-PLAN(speaker,hearer,laststep,newstep,`
	`nextstep,plan)`
Prerequisites:	`WANT(hearer,plan)`
	`LAST(laststep,plan)`
Decomposition-1:	`REQUEST(speaker,hearer,newstep)`
Decomposition-2:	`REQUEST(speaker,hearer,nextstep)`
Effect:	`STEP(newstep,plan)`
	`AFTER(laststep,newstep)`
	`AFTER(newstep,nextstep)`
	`NEXT(newstep,plan)`
Constraint:	`STEP(laststep,plan)`
	`STEP(nextstep,plan)`
	`AFTER(laststep,nextstep)`
	`AGENT(newstep,hearer)`
	`not CANDO(speaker,nextstep,plan)`
	`MODIFIES(newstep,laststep)`
	`ENABLES(newstep,nextstep)`

Figure 2.1: `correct-plan` plan

plan), and lets the speaker correct the underlying plan by introducing a new step into it. The constraints of `correct-plan` ensure that the new step is a variant of the last executed step, that the next step is not executable (presumably because the last step didn't enable some condition that the next step relied on), and that the new

step will enable the next step. The effect of `correct-plan` is that the new step will be inserted into the underlying plan after the last step and before the next step and that it will be executed next. This plan is achieved by the speaker either requesting the new step or requesting the next step (which can't be executed).

The clarification plans and the vocabulary for referring to plans provide a good starting point for modeling clarification subdialogues. However, a richer set of clarification plans and a corresponding richer vocabulary are needed to deal with clarifications in general. Litman and Allen's plans deal with only changing a single primitive action in an underlying plan where the change does not have any side effects in the rest of the plan and where the action is the current focus. However, these limitations must be relaxed. In general, a branch occurring anywhere in the plan tree should be able to be repaired even when it has side effects elsewhere in the plan.

Chapter 3

Referring Expressions

Searle (1969) proposed that referring is a communicative act. With the advent of using planning techniques to account for illocutionary speech acts, viewing referring as a communicative act allows planning techniques to account for its usage. However, previous plan-based models of referring (Cohen, 1984; Appelt, 1985a) have treated the process of determining the description (proposition content) as a primitive action.

This thesis proposes using planning techniques to both generate and understand the propositional content of referring expressions. In other words, the individual components of a description will be accounted for in the planning paradigm. This approach has several advantages. First, it allows both tasks to be captured in a single paradigm that is used for modeling general intelligent behaviour. Second, it allows more of the content of an utterance to be accounted for by a uniform process. Third, only a single knowledge source for referring expressions is needed instead of having this knowledge embedded in special algorithms for each task.

The outline of the rest of this chapter is as follows. We first discuss the scope of our work on referring expressions. Second, we define some predicates that will be used in our plan schemas. Third, we present the `refer` plan, the surface speech acts, and the subplans that are used in decomposing the `refer` plan into the surface speech acts. Fourth, we show how these plans are used for constructing and inferring referring expression plans.

3.1 Scope of our work

In order for a referring act to be successful, the referring expression must enable the hearer to identify the referent. For our work, we focus on references to objects that are mutually known by both the speaker and the hearer. Furthermore, we assume that identification is achieved by the hearer checking his beliefs as opposed to performing

physical actions, such as testing the contents of a glass to determine if it is a martini (Perrault and Cohen, 1981). This type of referring expression corresponds to Appelt's *shared concept activation with identification intention* (1985c).

When building a referring expression or when trying to identify a referent of a referring expression, the context in which it is uttered is very important. The context allows some objects to be referred to by pronouns or by definite descriptions that do not fully distinguish the object from all other objects that could be referred to (Grosz, 1981). But this thesis addresses the issue of referring to objects that are not in focus, and hence does not consider context. However, a primitive model of context could be incorporated into our work on the basis of sets of objects that are in focus. An interesting problem would be to integrate default inferences, sanctioned by the context, about what properties the referent can be assumed to have.

The beliefs of the speaker and hearer must also be taken into account. The propositional content of a referring expression does not express truths about the world but beliefs about the world. In constructing and inferring referring expression plans, our model of an agent takes into account its own beliefs and its beliefs about the other agent.

3.2 Vocabulary

Before we present the plan schemas for referring expressions, we need to introduce some vocabulary that is used in the plan schemas. We use the standard definitions of `goal`, `bel`, and `bmb`, and we introduce the predicate `bub`.

goal(Agt,Goal): Agent `Agt` has the goal `Goal`. An agent will act to make its goals true.

bel(Agt,Prop): Agent `Agt` believes proposition `Prop`.

bmb(Agt1,Agt2,Prop): Agent `Agt1` believes that it is mutually believed by itself and `Agt2` that proposition `Prop` is true. The standard definition of this predicate is given below:[1]

[1] In the plan schemas, the predicate `mb` is used; however, this predicate is always implicitly embedded inside the speaker's beliefs, thus giving `bmb`. This also applies to the predicate `bub`.

```
bmb(Agt1,Agt2,Prop)  ≡  bel(Agt1,Prop) and
                        bel(Agt1,bel(Agt2,Prop)) and
                        bel(Agt1,bel(Agt2,bel(Agt1,Prop))) and
                        etc[2]
```

bub(Agt1,Agt2,Prop): Agent `Agt1` believes that it is in the union of its beliefs and `Agt2`'s beliefs that proposition `Prop` is true. This predicate is similar to mutual belief except that, rather than an infinite conjunction of beliefs, it is defined as an infinite disjunction.

```
bub(Agt1,Agt2,Prop)  ≡  bel(Agt1,Prop) or
                        bel(Agt1,bel(Agt2,Prop)) or
                        bel(Agt1,bel(Agt2,bel(Agt1,Prop))) or
                        etc
```

This predicate is used to determine whether or not more modifiers should be added to the description, as will be explained in section 3.5.

3.3 Refer Plan

Since we are viewing referring as a communicative act, there is a plan schema for referring, shown in figure 3.1. (We adopt the Prolog convention that variables begin with an upper case letter, and all predicates and constants begin with a lower case letter.) The `refer` plan takes as a parameter a term representing the discourse

Header:	`refer(Entity)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Decomposition:	`s-refer(Entity)`
	`describe(Entity)`
Effect:	`bel(Hearer,goal(Speaker,knowref(Hearer,Entity)))`

Figure 3.1: `refer` plan

entity (Webber, 1983) of the referring expression that is being constructed. Such a discourse entity is represented by `entity(Var,Object)`, where `Object` represents the object being referred to, and `Var` is a unique identifier. The decomposition of the plan consists of the primitive action `s-refer`, which signals that the speaker is referring,

[2]Since this definition involves an infinite conjunction of beliefs, it is not possible to compute mutual belief in this way. Previous models have used rules for inferring mutual belief (Clark and Marshall, 1981; Perrault, 1990). We follow a similar approach (see section 7.4).

and the plan **describe**, which creates the description. These plans are presented later in the chapter. The predicates in the plan schema labeled by *where* are used to instantiate the variables **Speaker** and **Hearer** to either **system** or **user**.[3]

The effect of the **refer** plan is that the hearer will believe that the speaker has a goal of the hearer knowing the referent of the referring expression.[4] The effect has been formulated in this way so that it will be achieved by the hearer inferring the speaker's plan, regardless of whether the hearer is able to determine the referent. Following Litman and Allen (1987), we use the predicate **knowref** to represent that an agent knows a referent. However, our definition differs from theirs. They define **knowref(A,T,P)** as **A** knowing a description of term **T** that satisfies proposition **P**, whereas we define **knowref(A,E)** as **A** knowing the object that corresponds to the discourse entity **E**.

The effect shown in figure 3.1 is the only one included in any of the plan schemas for referring expressions. We view this as a simplification, for each of the subplans should have an effect stated for it. However, since we are chaining by decomposition and since we assume that plan inference always derives the correct plan derivation, nothing is lost by this simplification. Furthermore, determining the effects of each plan and developing a vocabulary to state them in is not a simple task.

Although we present this plan in isolation of a system that plans illocutionary speech acts, this plan is similar to Appelt's concept activation action (1985a) and Cohen's request to identify (1984) in terms of its communicative role, and hence could be integrated into their plan-based models of generation.

3.4 Surface Speech Acts

For referring expressions, we need a set of primitive actions. Since these primitive actions will correspond to parts of an utterance, they can be viewed as surface speech actions. The surface speech actions that we chose are such that each component of a description is accounted for by a separate action, thus enabling our model to account for the content of referring expressions in the planning paradigm.

3.4.1 s-refer

The **s-refer** action is performed by the speaker to signal to the hearer that she is referring to an object, and that she intends him to identify the object. This is a

[3]Our usage of *where*-predicates stems from the lack of functions and typed variables in Prolog.

[4]This effect does not capture that the speaker will believe that the referring expression enables the hearer to identify the referent. However, it will suffice for this thesis.

distinct action because a referring speech act is sometimes linguistically marked, for instance, when uttered in an appropriate context, "the bird, you know?", "see the bird," and even by the definite article.

Header:	`s-refer(Entity)`

Figure 3.2: `s-refer` plan

3.4.2 s-attrib

For each predicate that describes an object in terms of an attribute, there will be an `s-attrib` action schema.[5] The parameters of these schemas are a discourse entity, and a predicate that encodes the attribute. The predicate is encoded using lambda calculus since it allows the syntax of predicates to be standardized, thus simplifying other plan schemas that might need to make use of the predicate. The constraints of the plan specify that the speaker believes that the attribute is mutually believed to be true of the referent.[6] In figure 3.3, the schema is given for attributing colour.[7]

Header:	`s-attrib(Entity,λX·colour(X,Colour))`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
	`ref(Entity,Object)`
Constraint:	`mb(Speaker,Hearer,colour(Object,Colour))`

Figure 3.3: s-attrib plan

3.4.3 s-attrib-rel

The `s-attrib-rel` actions are similar to the `s-attrib` actions. However, instead of describing an object in terms of an attribute, the object is described relative to some other object. Hence, there are action schemas for relationships that can hold between objects. Note that when using `s-attrib-rel`, it is necessary to refer to the related object, which is done by the parent plan of `s-attrib-rel` (see section 3.5.4),

[5]Having multiple plan schemas with the same header corresponds to using an abstraction operator (Kautz and Allen, 1986), where the abstraction operator maps the plan header to each of the plan schemas.

[6]This constraint is too strong and could be weakened to take into account Perrault and Cohen's work on inaccurate reference (1981).

[7]The `ref` predicate in the plan schema maps a discourse entity representing a referring expression onto the object that is being referred to. It is used in order to access the beliefs about the referent.

and to create a discourse entity for the related object, which is done by the second `ref` predicate.[8] In figure 3.4, the schema for describing that an object is in another object is given.

Header:	`s-attrib-rel(Entity,LocEntity,`λ`X·`λ`Y·at(X,Y))`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
	`ref(Entity,Object)`
	`ref(LocEntity,Loc)`
Constraint:	`mb(Speaker,Hearer,at(Object,Loc))`

Figure 3.4: `s-attrib-rel` plan

3.4.4 s-attrib-rel-group

The `s-attrib-rel-group` actions are also similar to the `s-attrib` actions. However, in this case, the object is described relative to the set of objects that could satisfy the propositional content of the actions that have already been planned. For instance, it is used to describe that the object is the largest object of those that satisfy the rest of the description. For each predicate that allows an ordering and for each way it can be ordered, there is a separate action schema. In figure 3.5, the schema for saying that an object is the largest is given.

Header:	`s-attrib-rel-group(Entity,`λ`X·max(size,X),Cand)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
	`ref(Entity,Object)`
Constraint:	`mb(Speaker,Hearer,max(size,Object,Cand))`

Figure 3.5: `s-attrib-rel-group` plan

3.4.5 An Example

With these primitive actions, we can represent the surface speech acts associated with referring expressions. As an example, the surface speech acts for the referring expression "the black bird in the cage" are given below:

[8] It is unclear where a discourse entity should be created. We have taken the approach of making the `ref` predicate create new discourse entities if the variable representing the discourse entity is uninstantiated.

```
s-refer(entity(1,bird2))
s-attrib(entity(1,bird2),λX·category(X,bird))
s-attrib(entity(1,bird2),λX·colour(X,black))
s-attrib-rel(entity(1,bird2),entity(2,cage5),λX·λY·in(X,Y))
s-refer(entity(2,cage5))
s-attrib(entity(2,cage5),λX·category(X,cage)
```

3.5 Intermediate Plans

This section presents the subplans that are used in decomposing the **refer** plan into the surface speech acts, namely **describe**, **headnoun**, **modifiers**, and **modifier**. These plans encode the knowledge of how a description can allow a hearer to identify an object.

To ensure successful reference, the description that these intermediate plans build must enable the hearer to uniquely identify the referent. The intermediate plans reason about the success of the description through the use of *candidate sets*. A candidate set represents the set of objects that the speaker believes could satisfy the propositional content of the referring expression that has already been planned. Since the speaker and hearer are not assumed to have identical beliefs about the objects in the world, the speaker must consider the union of their beliefs (**bub**) in computing the candidate set in order to minimize the chance of an infelicitous reference.[9] Once a description has been planned in which the associated candidate set contains only the representation of the referent, the speaker will believe that the description will enable the hearer to identify the referent.

The candidate sets also play a crucial role in the plan inference process. When inferring the referent of a referring expression, the strategy whereby each component of the expression is used to constrain the choice of referent is captured in the plan schemas by the operations that derive the candidate sets. Hence, the hearer will be able to determine the candidate sets by evaluating the plan derivation that he has ascribed to the speaker. Assuming that the beliefs that the speaker has ascribed to the hearer are correct, then the candidate set corresponding to the entire expression will contain only the referent.

We now need to account for how candidate sets are built, and how these sets

[9]Consider the scenario in which the speaker wants to refer to **bird2**, which he believes is mutually believed to be black. Let's also assume that there is another bird that the speaker believes to be brown, but the speaker believes that the hearer believes it is black. By using the union of their beliefs in determining candidate sets, the speaker will find the description "the black bird" is potentially infelicitous and will therefore add another modifier.

will affect the rest of the referring expression. First, we propose the use of *mental actions*. Mental actions are similar to primitive actions, except that they do not have any observable effects. Instead, their effects are changes in the state of the planning agent. Second, we allow variable instantiation to have side effects, whereby a variable is instantiated in one step of a plan derivation and its value is used by another step. Thus, the mental actions are used to build the candidate sets, and these candidate sets can affect other steps in the plan derivation.

3.5.1 describe

The `describe` plan (figure 3.6) is one of the steps of the `refer` plan and is used to construct a description of the object. Its decomposition consists of `headnoun` and `modifiers`. The `headnoun` step instantiates the candidate set to the set of objects that matches the head noun that is chosen. This candidate set is passed to the `modifiers` step so that it will know which objects the referent must be distinguished from.

Header:	`describe(Entity)`
Decomposition:	`headnoun(Entity,Cand)`
	`modifiers(Entity,Cand)`

Figure 3.6: `describe` plan

3.5.2 headnoun

The plan `headnoun` (figure 3.7) has two steps. The first step is `s-attrib`, which

Header:	`headnoun(Entity,Cand)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
	`world(World)`
Decomposition:	`s-attrib(Entity,`λ`X·category(X,Category))`
	`subset(World,`λ`X·ub(Speaker,Hearer,`
	`category(X,Category)),Cand)`

Figure 3.7: `headnoun` plan

determines the head noun of the referring expression. The second step is the mental action `subset`, which determines the candidate set.[10] The candidate set is computed

[10]The predicate `subset(Set,Pred,Subset)` computes `Subset` such that the following holds: `Subset` $= \{X \mid X \in$ `Set` *and* `Pred`$(X)\}$. If `Subset` is empty, the mental action will fail.

by finding the subset of the objects in the world that the speaker believes could be referred to by the head noun.

3.5.3 modifiers

The `modifiers` plan attempts to ensure that the referring expression that is being constructed is believed by the speaker to allow the hearer to uniquely identify the referent. It accomplishes this by repeatedly choosing predicates that are true of the referent, until the conjunction of the chosen predicates apparently identifies the referent. Rather than extending the planning model to define the notion of iterative plans, we use recursion.

There are three plan schemas for `modifiers`. The first schema (figure 3.8) embodies the recursion. Its first step is `modifier`. This step, through its decomposition,

Header:	`modifiers(Entity,Cand)`
Decomposition:	`modifier(Entity,Cand,NewCand)`
	`modifiers(Entity,NewCand)`

Figure 3.8: `modifiers` plan

adds a component to the description and updates the candidate set by computing the subset of it that satisfies the new component. The updated candidate set is then passed to the second step, which is the recursive instantiation of `modifiers`.

The second `modifiers` schema (figure 3.9), which has a null decomposition, terminates the recursion.[11] Its constraint ensures that only one object matches the propositional content of the referring expression.

Header:	`modifiers(Entity,Cand)`
Where:	`ref(Entity,Object)`
Constraint:	`Cand = [Object]`
Decomposition:	`null`

Figure 3.9: `modifiers` plan

The third schema (figure 3.10) also terminates the recursion. It decomposes into the surface speech action `s-attrib-rel-group`, which adds a component to the description that distinguishes the referent from the other candidates.

[11] In order to distinguish this plan from a primitive action, it has a step that is marked `null`.

Header:	`modifiers(Entity,Cand)`
Decomposition:	`s-attrib-rel-group(Entity,Pred,Cand)`

Figure 3.10: `modifiers` plan

3.5.4 modifier

The `modifier` plan accounts for individual components of the description. There are two plan schemas for `modifier`, one corresponding to `s-attrib` and one corresponding to `s-attrib-rel`.

The first schema is shown in figure 3.11. Its decomposition consists of the surface speech act `s-attrib` and a mental action that determines the new candidate set (`NewCand`) by including only the objects from the old candidate set (`Cand`) for which the predicate could be believed to be true.

Header:	`modifier(Entity,Cand,NewCand)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Decomposition:	`s-attrib(Entity,Pred)`
	`subset(Cand,λX·ub(Speaker,Hearer,Pred(X)),`
	`  NewCand)`

Figure 3.11: `modifier` plan

The second schema (figure 3.12) has `s-attrib-rel` in its decomposition. Since this action describes an object in terms of some related object, the schema has a step to refer to it. It also has a step to update the candidate set.

Header:	`modifier(Entity,Cand,NewCand)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Decomposition:	`s-attrib-rel(Entity,OtherEntity,Pred)`
	`ref(OtherEntity,Other)`
	`subset(Cand,λX·ub(Speaker,Hearer,Pred(X)(Other)),`
	`  NewCand)`
	`refer(OtherEntity)`

Figure 3.12: `modifier` plan

3.6 Constructing the Simplest Referring Expression

The plan schemas described above underconstrain the choice of referring expressions. In particular, nothing in the plan schemas captures the preference of people to use minimal referring expressions. Since the content of an expression is being accounted for in the planning paradigm, this preference can be stated in more general terms as a heuristic for constructing plans: *prefer simple plans*[12]. This heuristic has intuitive appeal, and when planning with the intention that the hearer infer the plan, this heuristic is related to Grice's maxims of conversation.

In order to use the heuristic of preferring simple plans, it is necessary to state it in an operational way. There are many ways that this could be done. For our purposes, it suffices to prefer the plan derivation with the least number of primitive actions.[13] By using this planning heuristic, we can account for Ehud Reiter's results on avoiding conversational implicature (1990), without resorting to special algorithms that have no intuitive appeal.[14]

The plan schemas have been designed so that a plan can be constructed by chaining through decomposition only. This simplifies the plan construction process, for it eliminates the need to chain through preconditions. The plan constructor starts with a goal and finds the shallowest plan derivation that achieves this goal, such that the constraints are true and the mental actions are executable for each instance in the plan derivation.

3.6.1 An Example

Consider the scenario in which the system believes that there are three birds—a small turquoise one (`bird1`), a large black one (`bird2`), and a large white one (`bird3`)—plus some other non-bird objects in the world. Also suppose that the system (as the speaker) has the goal of referring to `bird2`.

The derivation in figure 3.13 shows the most preferred plan created by the plan constructor to satisfy the goal of referring to `bird2`. Each plan, primitive action, and mental action that is in the derivation is shown in a box, and the steps of a plan are shown as boxes inside of the box representing the plan. Any variables that

[12]By *simple plans* we mean plans that are minimal in some manner. We are not referring to Pollock's usage of *simple plan* (1990, p. 91).

[13]See section 7.2.1 for further comments on this planning heuristic.

[14]As a counterpoint, our approach does not guarantee that a referring expression can be found in polynomial time.

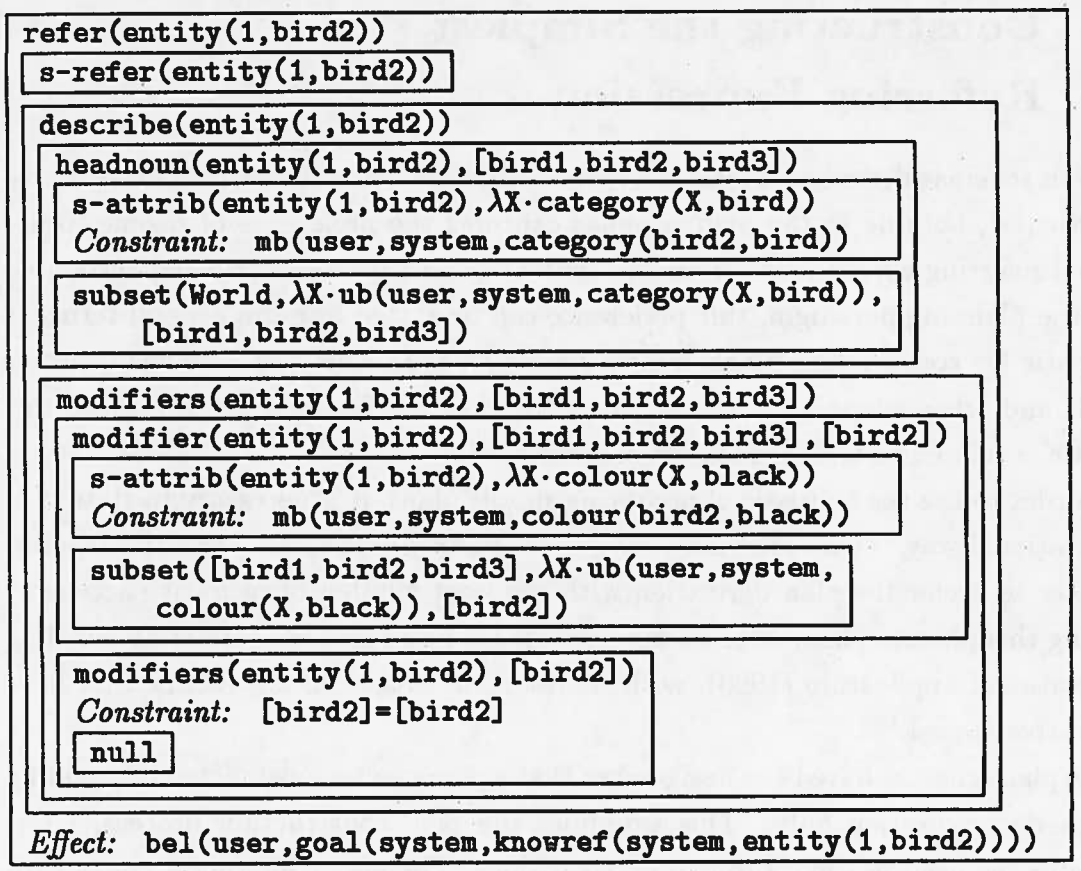


Figure 3.13: Constructed referring expression plan

are instantiated during the plan construction process are labeled by *instantiation*, and the effects of this instantiation on the rest of the plan derivation are shown by replacing the variable with its value. For instance, in instantiating the constraint `mb(system,user,category(bird2,Category))`, `Category` is instantiated to `bird` and this instantiation is carried through to the second step of `headnoun` as shown. The resulting surface speech acts of this derivation are listed below, and they correspond to the phrase "the black bird."

```
s-refer(entity(1,bird2))
s-attrib(entity(1,bird2),λX·category(X,bird))
s-attrib(entity(1,bird2),λX·colour(X,black))
```

3.7 Understanding a Referring Expression

To understand an utterances that he hears, the hearer must determine what the speaker intended by way of the utterance. In our model, this is accomplished by plan

inference. By inferring the speaker's plan, the hearer can determine the effects of the plan, which are assumed to be the speaker's intentions in making the utterance.

The plan inference process takes as input the surface speech acts returned by a parser. The first step of the process recognizes the plan derivation. It finds a plan that has as its yield the primitive actions that were observed. Motivated by Pollack's work on inferring invalid domain plans (1986), it does not ensure that the constraints and mental actions are satisfiable, thus allowing plans with unsatisfiable constraints or mental actions (invalid plans) to be recognized. The second step evaluates the plan to ensure that it is valid, and if it is not, the node that is not satisfiable is noted.[15]

The plan evaluator prefers to evaluate the constraints and mental actions in the order specified by the plan schemas. However, these schemas have been formulated for plan construction, and there is a difference in knowledge that the speaker has when constructing a plan and the knowledge that the hearer has when inferring it. Hence, it might not be efficient or even possible to evaluate in the order specified in the plan schemas. Thus, the constraints and mental actions are evaluated in an order that takes into account the hearer's knowledge and meta-level knowledge about the predicates in the plan schemas.

For referring expressions, once the plan inference process is completed, the hearer will have inferred that the speaker's plan was the `refer` plan and will have the belief that the speaker has a goal of the hearer knowing the referent of the referring expression. If the plan evaluation was successful, the hearer will in fact know the referent, since the variable representing the referent will have been instantiated. Otherwise, if it wasn't successful, one of the constraints or mental actions in the inferred plan derivation will be found to be unsatisfiable, which could result in the hearer being unable to identify the referent.

3.7.1 An Example

Consider again the scenario in which the system (as the hearer) believes that there are three birds, a small turquoise one (`bird2`), a large black one (`bird2`), and a large white one (`bird3`), plus some other objects in the world. Suppose that the user has uttered "the black bird" and that a parser has derived the primitive actions listed below:

[15] In a general model of plan inference, it might not always be necessary to evaluate a recognized plan. The decision might depend on the effects of the inferred plan, and on whether there are several completing plan derivations that can account for the primitive actions that were observed.

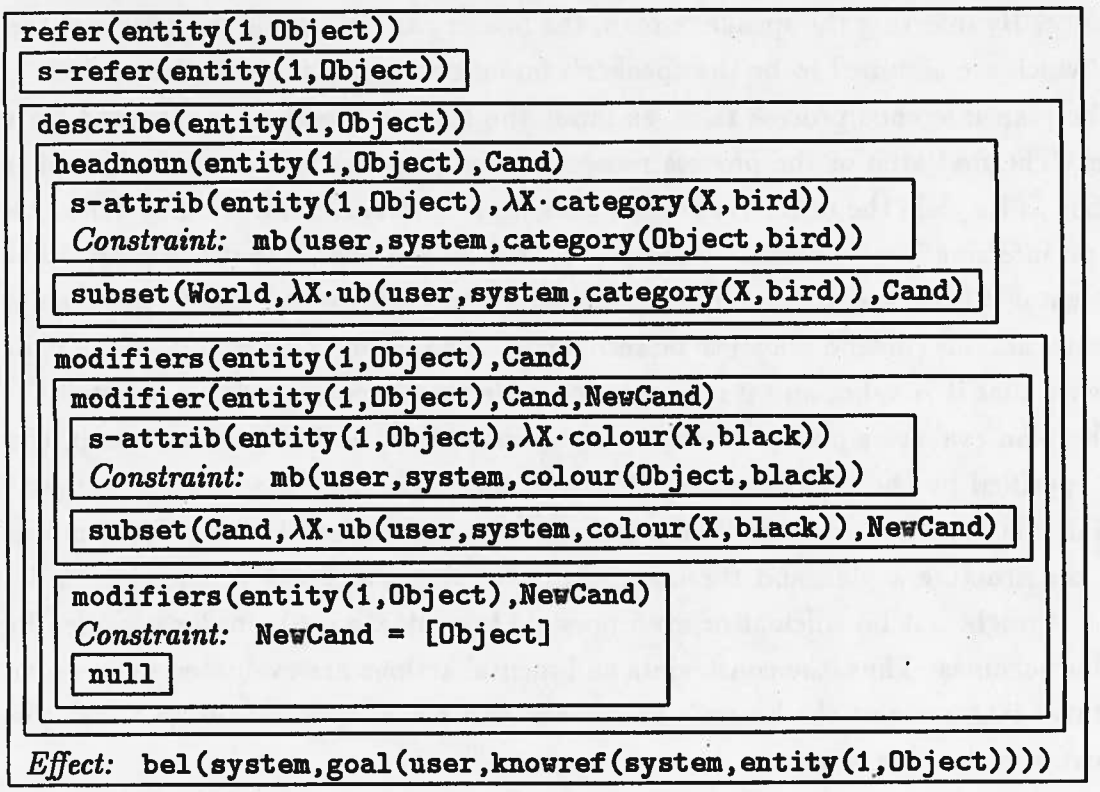


Figure 3.14: Recognized referring expression plan

```
s-refer(entity(1,Object))
s-attrib(entity(1,Object),λX·category(X,bird))
s-attrib(entity(1,Object),λX·colour(X,black))
```

In order to understand the user's utterance, the primitive actions are used as input to the plan inference process. The first step of the plan inference process will recognize the user's plan, resulting in the derivation shown in figure 3.14 (with the where-predicates instantiated). Note that all variables that co-refer have been unified.[16] For instance, the variables for the candidate sets in the `describe`, `headnoun`, `modifiers`, and `modifier` plans have been unified. Unifying co-referring variables also has the effect that all variables that co-refer to a variable that has been instantiated in the primitive action will be instantiated.

The second step of the plan inference process evaluates the constraints and mental actions in the plan derivation. For this plan derivation, the evaluator will first instantiate the mental action `subset` in the headnoun plan. This will result in `Cand` being instantiated to `[bird1,bird2,bird3]`. Second, the `subset` action in the `modifier`

[16]The variable names have been changed for readability. In reality, Prolog gives all variables arbitrary names such as _34.

plan is evaluated yielding `Cand1` being instantiated to `[bird2]`. Third, the constraint `Cand1 = [Object]` in the second instance of the **modifiers** plan is evaluated, yielding `Object` being set to `bird2`. Once this instantiation is made, the rest of the plan derivation can be evaluated in any order. The resulting plan derivation is the same as the one shown in figure 3.13 except that **system** and **user** are exchanged. So, as can be seen, the stated effect of the **refer** plan will have the variable `Object` instantiated to `bird2`, and so the system has deduced the referent of the referring expression.[17]

3.8 Summary

In this chapter, we presented our model, which is based on the planning paradigm, of how the propositional content of referring expressions can be generated and understood. Although we have not covered all types of referring expressions, we feel that our model can be extended to handle a large number of them. In particular, we feel that this approach can incorporate non-linguistic actions that are used in conjunction with referring expressions, such as pointing at a referent. As well, descriptions sometimes specify a plan of actions for the hearer to perform to identify the referent. Since our approach is based on planning, it could offer a solution to how such referring expressions are constructed and understood.

[17] In chapter 5, we will give rules that the system can apply so as infer that it knows the referent given the knowledge it has inferred from the plan derivation.

Chapter 4

Clarifications

Clark and Wilkes-Gibbs (1986) present a model of how conversational participants collaborate in making a referring action successful. Their model consists of conversational moves that express a judgement of the success of a referring expression and conversational moves that refashion an expression. However, their model is not computational. They do not account for how the judgement is made, how the judgement effects the refashioning, nor the content of the moves.

This chapter casts Clark and Wilkes-Gibbs's model into a computational framework based on the planning paradigm. The conversational moves of their model are accounted for by discourse plans. This approach allows plan construction to be used for generating a clarification[1] and plan inference for understanding a clarification.

The discourse plans, however, only provide part of the solution. We must still account for how a plan that is inferred from an utterance influences how an agent choices the goal it tries to achieve in the next utterance. This part of our model is presented in the next chapter, allowing us in this chapter to concentrate on the construction and inference of clarifications in isolation.

The outline of the rest of this chapter is as follows. We first account for how referring expressions are judged. Second, we present a vocabulary that our discourse plans use to reason about and manipulate a plan derivation that corresponds to a referring expression. Third, we present the discourse plans and the surface speech acts that we employ. Fourth, we give examples of how clarifications are constructed and inferred in our model.

[1] We use the term *clarification* since the conversational moves of judging and refashioning a referring expression can be viewed as clarifying the referring expression.

4.1 Judging Referring Expressions

Clark and Wilkes-Gibbs propose a number of conversational moves that participants use in collaborating to make a referring action successful. They claim that after a referring expression (or a refashioned referring expression) is presented, the hearer judges it and that he either accepts it, rejects it, or postpones the judgement. If he rejects it or postpones the judgement then either he or the speaker will refashion it by replacing it or expanding it. This process is repeated until a final referring expression is mutually accepted.[2]

Clark and Wilkes-Gibbs, however, do not account for why conversational participants choose to accept or reject a referring expression or postpone the judgement. Nor do they account for how a participant chooses whether to replace a referring expression or expand it. In order to cast their model of conversational moves into a computational framework, we need to account for how these decisions are made.

We propose that the hearer judges the referring expression by determining whether any objects match the description. Either exactly one object matches, no objects match, or multiple objects match the referring expression. If exactly one object matches the referring expression, then the hearer will accept it. If he doesn't find any objects that match it, then the expression overconstrains the choice of referent and he will be unable to identify the referent, and will thus reject the expression or part of it. If he finds multiple objects that match the expression then it underconstrains the choice of referent and although he finds it acceptable so far, he would postpone his judgement.

If the referring expression was rejected or the judgement was postponed, then either the hearer or the speaker will refashion it. If the hearer found that the referring expression overconstrained the choice of referent, then the expression, or part of it, will need to be replaced. If the hearer found that it underconstrained the choice of referent, then the expression will either be expanded, or part of it replaced. Whether this is done by the hearer will depend on whether he knows the referent (if he is the initiator of the referring expression) or he is able to choose a plausible candidate as the referent.

Since we are using the planning paradigm to account for the generation and understanding of the propositional content of referring expressions, we can cast the judgement in terms of the evaluation of the inferred referring expression plan. When the hearer tries to understand the referring expression, he employs plan recognition

[2]In this thesis, we do not incorporate Clark and Wilkes-Gibbs's work on how speech errors are repaired.

and then evaluates the recognized plan. If the evaluation is successful, then he will have identified the referent of the referring expression. If it is not, then the hearer will have identified a constraint or mental action that is not satisfiable. The node in the plan derivation at which the evaluation failed captures the information needed to decide whether the referring expression is overconstrained or underconstrained. If it is overconstrained, then one of the constraints on an `s-attrib`, `s-attrib-rel`, or `s-attrib-rel-group` speech act will have been violated.[3] If it is underconstrained, then the constraint on the instance of the `modifiers` plan that terminates the recursion and that has a null decomposition will have been violated. Hence, we use structural properties of where the violation occurred in the plan derivation to determine whether the referring expression overconstrains or underconstrains the choice of referent, which in turn determines whether the referring expression is rejected or the judgement is postponed.

4.2 Vocabulary

In order to model how agents collaborate in making referring expressions, we need a vocabulary for reasoning about plans, reasoning about the parts of a plan, and for refashioning plans. The vocabulary that we present is motivated by work done by Litman and Allen (Litman, 1985; Litman and Allen, 1987) in understanding clarification subdialogues. The vocabulary is not intended to be complete, but only to allow us to formalize the conversational moves involved in clarifying a referring expression.

4.2.1 Referring to Plans

The predicates defined in this section are used to refer to referring expression plans and to reason about their success or failure in achieving a goal.

plan(Agt,Plan,Goal): `Agt` has a plan derivation `Plan` for achieving `Goal`.

achieve(Plan,Goal): Doing the primitive actions in the plan derivation `Plan` will cause `Goal` to be true.

[3]The evaluation could also have failed on the mental action of the `modifier` plan that updates the candidate set. This occurs only because the evaluator dynamically chooses the order in which to evaluate the plan derivation and might choose to evaluate the mental action in the `modifier` plan before evaluating the constraint on the surface speech act. So, the evaluator, when it finds an error, tries to determine if the error is a result of a constraint or mental action whose evaluation was postponed. Our algorithm for determining this looks for syntactic similarities. A better approach would reason about the interactions of constraints and mental actions in the plan derivation.

error(Plan,Node): The plan derivation `Plan` has an error at the node named `Node`. Nodes in the plan derivation are given names in order to distinguish two nodes that have the same content.

replace(Plan,NewPlan): The plan derivation `NewPlan` will achieve the goal that `Plan` is supposed to achieve.[4]

4.2.2 Referring to Parts of a Plan

The following predicates are used to refer to parts of a plan derivation. Their primary usage in this thesis is to test for structural properties of plan derivations in order to determine whether the referring expression overconstrains or underconstrains the choice of referent.

content(Plan,Node,Content): In the plan derivation `Plan`, the node named by `Node` has content `Content`.

constraint(Plan,Parent,Constraint): In plan derivation `Plan`, the node `Parent` is a subplan and the node `Constraint` is a constraint of it.

yield(Plan,Node,Actions): In plan derivation `Plan`, the primitive actions in the subtree rooted at `Node` is `Actions`.

4.2.3 Manipulating and Evaluating Plans

In order to refashion a referring expression plan, we need to be able to construct a subplan that can be substituted into the existing plan, so that the resulting plan does not have any violated constraints or mental actions. This section presents the mental actions that accomplish these tasks.

construct(Goal,Plan,Actions): Invokes the plan constructor to construct a plan derivation `Plan` that achieves `Goal`. `Actions` are the primitive actions of the constructed plan derivation.

substitute(Plan,Node,NewPart,NewPlan,Actions): In plan derivation `Plan`, substitutes the content of `Node` by `NewPart` resulting in the plan `NewPlan` and the new primitive actions `Actions`. `NewPart` might be a constraint, in which case `Actions` are those that are changed as a result of the substitution.

[4] Although this predicate, which is similar to Litman's `replace(Stack,OldStack)` predicate (1985, p. 33), has the same name as one of Clark and Wilkes-Gibbs's conversational moves, it is used in stating the effects of both the *replace* and *expand* conversational moves.

evaluate(Plan): Evaluates the plan derivation `Plan`. This action succeeds if and only if the plan is valid.[5]

4.3 Discourse Plans

We propose using discourse plans to account for Clark and Wilkes-Gibbs's conversational moves in collaborating to make a referring action successful. These discourse plans are motivated by the work done by Litman and Allen (Litman, 1985; Litman and Allen, 1987) in understanding clarification subdialogues. By using discourse plans, both the generation and understanding of the conversational moves can be accounted for. These discourse plans are meta-plans that take as a parameter a plan derivation corresponding to a referring expression and use the predicates defined in section 4.2 to reason about and manipulate the derivation. The objective of this section is not to provide complete coverage of how participants can clarify a referring expression, but to demonstrate the feasibility of this approach. Although we focus on clarifying referring expressions, we feel that our approach can be generalized to handle other clarification subdialogues as well.

4.3.1 accept-plan

The discourse plan `accept-plan` (figure 4.1) is used by the speaker to establish the

Header:	`accept-plan(Plan)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`achieve(Plan,Goal)`
	`plan(Hearer,Plan,Goal)`
Decomposition:	`s-accept(Plan)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer,` `achieve(Plan,Goal))))`

Figure 4.1: `accept-plan` plan

mutual belief that a plan will achieve its goal. Since the speaker cannot directly affect the beliefs of the hearer, the effect of the schema is that the hearer will believe that the speaker has the goal that it be mutually believed that the plan achieves its goal. The constraints of the schema specify that the plan being accepted achieves its goal and the decomposition is the surface speech action `s-accept`.

[5]This predicate is treated as a mental action, in order to avoid the use of *post-constraints*.

4.3.2 postpone-plan

The discourse plan **postpone-plan** is used by the speaker if the referring expression plan underconstrains the choice of referent. The speaker uses this schema in order to tell the hearer that the plan is invalid and to inform him which node the evaluation failed at.

In figure 4.2, the plan schema for **postpone-plan** is given. The constraints specify that the violation occurred on the constraint of an instance of the **modifiers** plan that

Header:	`postpone-plan(Plan)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`error(Plan,ErrorNode)`
	`plan(Speaker,Plan,Goal)`
	`constraint(Plan,ParentNode,ErrorNode)`
	`content(Plan,ParentNode,ParentContent)`
	`yield(Plan,ParentNode,[])`
	`ParentContent = modifiers(Entity,Cand)`
Decomposition:	`s-postpone(Plan)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer,` `error(Plan,ErrorNode))))`

Figure 4.2: **postpone-plan** plan

has no primitive actions in its decomposition. In other words, the violation occurred on an instance of the **modifiers** plan that terminates the addition of **modifier** plans, and so it only applies if the referring expression is underconstrained.

The decomposition of this schema includes the **s-postpone** surface speech action. By observing this action, the hearer will be able to recognize that the speaker's plan is an instance of **postpone-plan**, and by evaluating the plan, he should be able to infer that the speaker thinks that the node in violation is the constraint on the terminating instance of **modifiers**. This knowledge will provide context for the subsequent refashioning.

As we have just explained, through plan inference, the hearer should be able to infer which node is in violation. In fact, this is the goal that the speaker is trying to achieve by this plan. However, the speaker cannot be certain that the hearer will be able to determine which node is in violation or whether he will agree that there is an error at the node. So, the stated effect of the schema is that the hearer will recognize the speaker's goal of the speaker and hearer mutually believing that there is an error at some node in the referring expression plan.

4.3.3 reject-plan

The discourse plan `reject-plan` is similar to `postpone-plan` in that it is also used by the speaker to tell the hearer that a referring expression plan is invalid and to inform him which node the evaluation failed at. In this case, however, it is used if the plan overconstrains the choice of referent.

The plan schema is given in figure 4.3. The effect of this plan is the same as the effect of `postpone-plan`. Its constraints specify that an error occurred on a constraint

Header:	`reject-plan(Plan)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`error(Plan,ErrorNode)`
	`plan(Hearer,Plan,Goal)`
	`constraint(Plan,ParentPlan,ErrorNode)`
	`yield(Plan,ParentPlan,Acts)`
	`length(Acts,1)`
Decomposition:	`s-reject(Plan,Acts)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer,` `error(Plan,ErrorNode))))`

Figure 4.3: `reject-plan` plan

of a plan that has a single primitive action in its yield. An alternative way of stating the constraints, which is less general, would be to specify that an error occurred on a constraint of a plan in which the plan is an instance of `s-attrib`, `s-attrib-rel`, or `s-attrib-rel-group`.

The decomposition of the schema consists of the `s-reject` surface speech action, which takes as a parameter the surface speech action that is in the yield of the plan that has the constraint violation. By observing the `s-reject` action, the hearer will be able to recognize that the speaker's plan is an instance of `reject-plan`, and by evaluating this plan, he should be able to determine the constraint that was violated.[6] This information will provide context for the subsequent refashioning of the referring expression.

[6]The reason the hearer should be able to determine the constraint is because the only plan instances that have a yield of a single surface speech action and that have a constraint are the surface speech actions `s-attrib`, `s-attrib-rel`, and `s-attrib-rel-group`, and these actions have exactly one constraint. In a more general plan library, it might not be possible for the hearer to determine the constraint or mental action that the speaker found unsatisfiable. However, it might not be necessary for the hearer to do so. The hearer might just need to know that there was an error in evaluating some constraint or mental action, where the constraint or mental action is influenced by the primitive action that was rejected.

4.3.4 expand-plan

The discourse plan **expand-plan** is used by the speaker to replace a referring expression plan with a new plan. The new plan is arrived at by adding extra components to the referring expression. Since our plan library is based on chaining by decomposition, this can only be done by constructing a subplan to replace a branch of the derivation that has a null yield. For our set of referring expression plan schemas, this corresponds to replacing the terminating instance of the recursive plan **modifiers** that has a null yield with a subplan that has a non-empty yield.

The plan schema for **expand-plan** is shown in figure 4.4. Its constraints are the same as for **postpone-plan**, and its effect is that the hearer will believe that the

Header:	`expand-plan(Plan)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`error(Plan,ErrorNode)`
	`plan(Hearer,Plan,Goal)`
	`constraint(Plan,ParentNode,ErrorNode)`
	`content(Plan,ParentNode,ParentContent)`
	`yield(Plan,ParentNode,[])`
	`ParentContent=modifiers(Entity,Cand)`
Decomposition:	`member(Object,Cand)`
	`ref(Entity,Object)`
	`construct(modifiers(Entity,Cand),Expansion,Acts)`
	`substitute(Plan,ParentNode,Expansion,NewPlan,Acts)`
	`s-actions(Plan,Acts)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer,` `replace(Plan,NewPlan))))`

Figure 4.4: **expand-plan** plan

speaker has the goal that the speaker and hearer mutually believe that a new plan replaces the current referring expression plan.

The decomposition of the schema specifies how a new referring expression plan can be built that will distinguish one of the objects that the current referring expression matches from the rest. The first step, `member(Object,Cand)`, chooses one of the objects. If the speaker is the initiator then she will know what the referent is and hence there is no choice involved in satisfying the mental action. However, if the speaker is the responder, then she will not know the referent, and hence will have to choose which object to suppose as the referent. In this case, she should use heuristics to choose a plausible referent, such as the object which has the greatest number of standard defaults. However, for this thesis, we will let the plan constructor arbitrarily

choose a member of the set of candidates. The third step[7] is used to construct an expansion that will distinguish the chosen candidate from the rest. The fourth step substitutes the expansion into the current referring expression, resulting in the refashioned referring expression **NewPlan**. The fifth step is the surface speech action **s-expand**, which is used to inform the hearer of the surface speech actions, **Acts**, that are being added to the referring expression plan.

An interesting scenario arises when the hearer has postponed judgement on the speaker's referring expression, and the speaker is now planning an expansion. In this case, although the speaker knows that the hearer thinks there is an error with the constraint of the terminating instance of the **modifiers** plan, she will not know which objects the hearer cannot distinguish the referent from, and since the speaker is examining her own plan derivation, she will think it doesn't have to be distinguished from any objects (since **Cand** contains only a single term, which represents the referent). Thus, when she constructs an expansion, the trivial plan derivation consisting of an instance of the **modifiers** plan with a null yield will be returned. To solve this problem, we require **construct** to always find a plan with at least one primitive action.[8]

4.3.5 replace-plan

The discourse plan **replace-plan** is used by the speaker to replace a referring expression plan with a new plan. The new plan is arrived at by replacing some of the primitive actions in a plan with new actions.

In this section we present two plan schemas for **replace-plan**. The first schema (figure 4.5) is similar to the **expand-plan** schema shown in figure 4.4. Instead of replacing a branch of a derivation that has a null yield, it replaces a branch whose yield is the surface speech action that caused the plan evaluator to fail.

One point that is worth explaining is the use of the **evaluate** action. This action succeeds if and only if the refashioned plan, **NewPlan**, is valid. In other words, it ensures that **NewPlan** actually identifies a unique object. This action is necessary because the **modifier** instance that is being replaced, **ModifierContent**, might not be the last **modifier** instance in the plan derivation. So, there might be

[7] We have intentionally skipped the second step because it is an artifact resulting from the manner in which discourse entities are treated. In the case where the speaker is the initiator, the discourse entity will already have its object parameter instantiated and so this step will ensure that **member** chooses the referent. If the speaker is the responder, then this step will update the discourse entity with the value of the chosen object.

[8] Since construct doesn't take into account what has already been planned, we still have the problem that it might choose a surface speech act that is already in the existing plan derivation.

Header:	`replace-plan(Plan)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`error(Plan,ErrorNode)`
	`plan(Hearer,Plan,Goal)`
	`constraint(Plan,ParentNode,ErrorNode)`
	`step(Plan,ModifierNode,ParentNode)`
	`content(Plan,ModifierNode,ModifierContent)`
	`ModifierContent = modifier(Entity,Cand,Cand1)`
Decomposition:	`member(Object,Cand)`
	`ref(Entity,Object)`
	`construct(modifier(Entity,Cand,Cand1),Expansion,Acts)`
	`substitute(Plan,ModifierNode,Expansion,NewPlan,Acts)`
	`evaluate(NewPlan)`
	`s-actions(Plan,Actions)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer,` `replace(Plan,NewPlan))))`

Figure 4.5: `replace-plan` plan

constraints that `Object` should satisfy that are not captured by `Cand`, resulting in `member(Object,Cand)` choosing an object that does not satisfy the additional constraints.

The second plan schema (figure 4.6), takes a different approach. This schema tries to relax the constraint of the plan that is the cause of the error, and using the relaxed constraint to refashion the referring expression. This approach is based on the heuristics that Goodman (1985) uses to relax a referring expression that is overconstrained. Our approach is more simplistic than Goodman's, for we allow only one constraint to be relaxed. Although this means that we cannot handle referring expressions that need more than one constraint relaxed, we do not feel that this is a serious limitation. In such cases it is probably best not to presuppose that the referring expression can be relaxed, but to rely on the hearer to try to repair it, or to use a different strategy to refashion it, such as the schema given in figure 4.5.

The decomposition of the schema is as follows. The first step is `relax-constraint`. As is explained in the next section, there might be many ways that a constraint can be relaxed, and some of these, when substituted into the referring expression plan, might not result in a plan that identifies exactly one object. Since the plan constructor arbitrarily chooses one, the mental action `evaluate` in the third step is used to ensure that the choice results in a referring expression plan that is valid. If the resulting referring expression plan is not valid, then this step fails, and so this choice

Header:	`replace-plan(Plan)`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`error(Plan,ErrorNode)`
	`plan(Hearer,Plan,Goal)`
	`content(Plan,ErrorNode,ErrorContent)`
Decomposition:	`relax-constraint(ErrorContent,NewContent)`
	`substitute(Plan,ErrorNode,NewContent,NewPlan,Acts)`
	`evaluate(NewPlan)`
	`s-actions(Plan,Acts)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer,` `replace(Plan,NewPlan))))`

Figure 4.6: `replace-plan` plan

for the relaxed constraint is rejected by the plan constructor. Finally, in the fourth step, the surface speech actions that are the result of the refashioning are conveyed to the hearer through the `s-actions` action.

relax-constraint

The `relax-constraint` schemas incorporate the heuristics that are used by Goodman (1985) in relaxing a description. These schemas take as input a constraint and return a constraint that is the result of weakening the original. For instance, the constraint that it is mutually believed that an object is turquoise could be relaxed to the constraint that it is mutually believed that it is green or to the constraint that it is mutually believed that it is blue.[9] In figure 4.7, a `relax-constraint` schema is given for relaxing a colour to a similar colour.

A problem with relaxing a constraint is that the relaxed constraint might not be believed by both participants. So, the agent that doesn't believe it will find the plan invalid, even though it knows that the violated constraint can be relaxed. A solution to this problem is to allow the belief system to automatically relax propositions. However, we feel that such relaxations should be explicitly sanctioned. So, the effect of the `relax-constraint` plan is to sanction the belief system to make the relaxation,[10]

[9]Since the new constraint will be used to refashion the referring plan derivation, it must be unifiable with the constraint of the schema of the primitive action. Hence we do not relax the constraint that it is mutually believed that the object is turquoise to the constraint that it is turquoise, green, or blue.

[10]Another solution to the problem would be to change the operator on the constraint from mutually believed to speaker believes. However, this constraint would not be unifiable with the plan schema of the primitive action.

Header:	`relax-constraint(mb(Agt1,Agt2,colour(Object, Colour)),mb(Agt1,Agt2,NewProp))`
Where:	`speaker(Speaker)`
	`hearer(Hearer)`
Constraint:	`similar(category,Colour,NewColour)`
Decomposition:	`NewProp=cat(Object,NewColour)`
Effect:	`bel(Hearer,goal(Speaker,mb(Speaker,Hearer, relax(colour(Object,Colour),NewProp))))`

Figure 4.7: `relax-constraint` plan

which is captured by the `relax` predicate:

relax(Prop,NewProp): If `NewProp` is believed, `Prop` can be inferred.

4.4 Surface Speech Acts

In this section, we present the surface speech acts that are used by the discourse plans. We have taken the approach that there are surface speech acts for accepting and clarifying plans. These surface speech acts take as a parameter the plan that is being accepted or clarified, and, depending on the speech act, a set of surface speech acts.

The surface speech acts that we propose are stated without constraints or effects. This is because their constraints and effects have already been captured by the constraints and effects of the discourse plans whose decomposition they are a part of. Since we assume that plan inference will always derive the correct plan derivation, nothing is lost by this simplification. In fact, the discourse plans `accept-plan`, `reject-plan`, and `postpone-plan` only have a single step in their decomposition, which is their corresponding surface speech act, and hence they could have been formulated as surface speech acts.

4.4.1 s-accept

The surface speech action `s-accept` (figure 4.8) is the only step in the `accept-plan`

Header:	`s-accept(Plan)`

Figure 4.8: `s-accept` plan

schema. It takes as a parameter a valid plan, and it can be linguistically realized by an explicit acknowledgement, such as "yes" or "okay" (Clark and Brennan, 1990).

4.4.2 s-postpone

The surface speech action **s-postpone** (figure 4.9) is the only step in **postpone-plan**. It takes as a parameter an invalid plan, and it can be linguistically realized by a tentatively voiced "okay."

Header:	s-postpone(Plan)

Figure 4.9: **s-postpone** plan

4.4.3 s-reject

The surface speech action **s-reject** (figure 4.10) is the only step in **reject-plan**. It takes as a parameter an invalid plan and a set of surface speech acts that are a

Header:	s-reject(Plan,Acts)

Figure 4.10: **s-reject** plan

subset of the primitive actions of the plan. Below is an example of an instance of this action, which can be linguistically realized by "What brown one?"

s-reject(Plan,[s-attrib(Entity,λX$\cdot$colour(X,brown))])

4.4.4 s-actions

The surface speech action **s-actions** (figure 4.11) is a step in both **replace-plan**

Header:	s-actions(Plan,Acts)

Figure 4.11: **s-actions** plan

and **expand-plan** and it signals that new primitive actions are being added to a referring expression plan in order to refashion it. Although two different kinds of surface speech actions could have been defined, one corresponding to **replace-plan** and one to **expand-plan**, we felt that this distinction is not always linguistically marked. Even when it is linguistically marked, the distinction might be due to the surface speech action being used in combination with either **s-reject** or **s-postpone**. Below is an example of this action:

s-actions(Plan,[s-attrib(Entity,λX$\cdot$colour(X,black))])

This could be linguistically realized by "the black one" or, if used in combination with `s-reject`, it could be realized by "don't you mean the black one?"

4.5 Constructing a Clarification

In constructing a clarification, we use the same algorithm as is used in constructing referring expression plans. However, the plan constructor is augmented to handle the mental actions that are given in section 4.2.3. In particular, for the mental action `construct`, it invokes itself, and for `evaluate`, it invokes the plan evaluator.

4.5.1 An Example

Consider the scenario where the user has uttered "the green bird" but the system only knows of a turquoise bird (`bird1`), a black bird (`bird2`), and a white bird (`bird3`). So, the system will have inferred the invalid plan derivation `p1` given in figure 4.12[11] and will have the following beliefs:

```
plan(user,p1,knowref(system,entity(1,Object)))
error(p1,p24)
```

Furthermore, assume that the system has already informed the user that plan `p1` is invalid, and that it is now trying to replace the plan by a new plan that will achieve the `knowref` goal. So, it will have the following goal:

```
bel(user,goal(system,mb(system,user,replace(p1,NewPlan))))
```

To achieve the goal, the plan constructor builds the derivation shown in figure 4.13, which is based on the second `replace-plan` schema (figure 4.6). However, this is not the only derivation that the planner considers. The plan constructor tries using the `postpone-plan` schema as the root of the derivation. But this choice is rejected because the constraint that the parent of the error node must not have any primitive actions in its yield is not true. The plan constructor also tries using the first `replace-plan` schema (figure 4.5) as the root. This results in a derivation with the same number of primitive actions as the chosen derivation. However, it is rejected due to an arbitrary decision on the part of the planner.[12]

[11] We have labeled only the plan derivation and the node that is in error with their node names. In reality, all nodes have node names.

[12] The constructor's choice was actually a good choice because a derivation based on the first schema involves the plan constructor making an arbitrary decision for the likely referent, whereas for the second schema, its decision is based on relaxing a constraint.

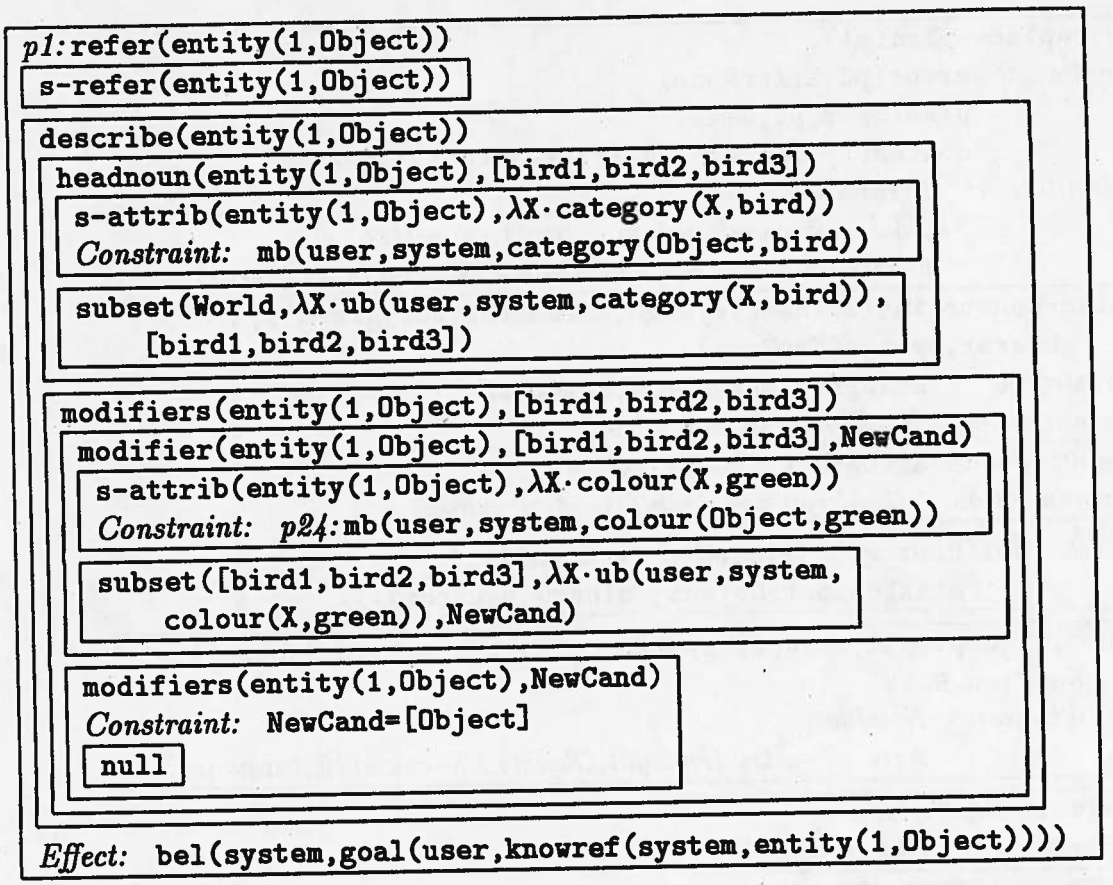


Figure 4.12: Inferred Referring Expression Plan (p1)

The derivation given in figure 4.13 shows how the plan constructor builds the chosen derivation. The text that is prefixed by *Instantiation* shows how the variables are instantiated.[13] From the constraints of `replace-plan`, `ErrorNode` is set to p24, and `ErrorContent` is set to `mb(user,system,colour(Object,green))`.[14] Next, the plan constructor proves the constraints of the steps in the decomposition. The first step is the `relax-constraint` plan. Proving its constraint results in `NewColour` being instantiated to `turquoise`.[15] The next step is the action `substitute`. This results in the refashioned referring expression plan p57 shown in figure 4.14, which, when evaluated in the next step, results in `Object` being instantiated to `bird1`. The last

[13]We do not show the value for `NewPlan` due to space and formatting considerations. However, we do indicate where it is instantiated.

[14]In our notation, we are not being careful about the scope of the existential quantifier for the variable `Object`.

[15]The plan constructor actually pursues every possible value that `NewColour` can be instantiated to; however, it is only the value `turquoise` that will allow the `evaluation` step to be proved.

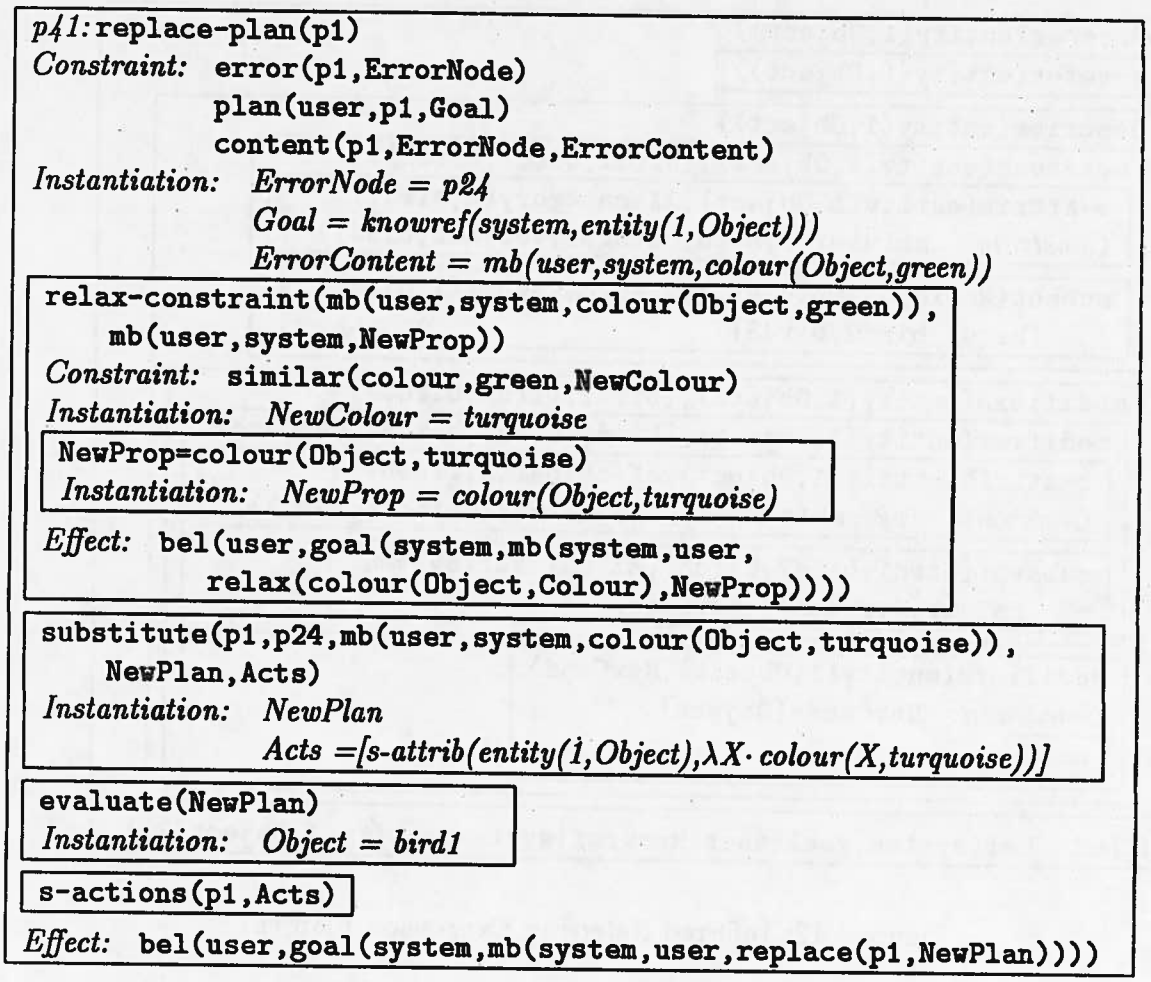


Figure 4.13: Constructed Refashioning Plan (p41)

step is the surface speech act `s-actions`, which is instantiated to the following:

`s-actions(p1,[s-attrib(entity(1,bird1),λX·colour(X,turquoise))])`

This action and the action resulting from informing the user that the original plan is invalid, could result in: "The green one? Don't you mean the turquoise one?"

4.6 Understanding a Clarification

In understanding a clarification, we use the same algorithm as is used in understanding a referring expression. However, a few extensions are needed to handle the mental actions `construct` and `evaluate`. When evaluating `construct`, the evaluator knows the primitive actions that `construct` must account for and the goal it is trying to achieve. So, the evaluator invokes the plan recognizer rather than the plan constructor.

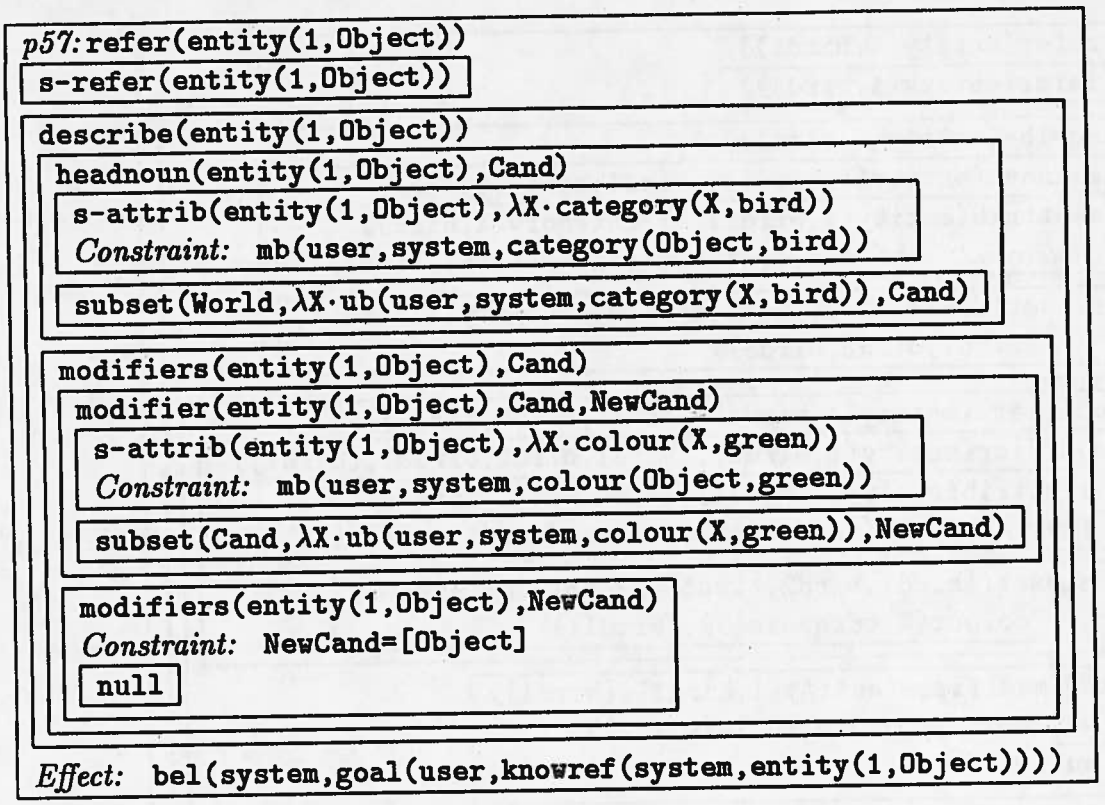


Figure 4.14: Refashioned Referring Expression Plan (p57)

When the mental action **evaluate** is in an inferred plan, it means that the speaker found the plan that is the parameter of the **evaluate** to be valid. However, the speaker might not intend that the hearer should find the plan to be valid, and so the parameter of the **evaluate** action is not evaluated. As is discussed in the next chapter, this view lets us distinguish between a clarification plan being valid versus the refashioned referring expression plan being valid, for the validity of the refashioned referring expression plan does not affect the validity of the clarification plan.

The remaining question to be answered is, if the **evaluate** mental action does not cause the hearer to evaluate the refashioned plan, what causes him to do so? We claim that the hearer decides to evaluate it by reasoning about the effects of the refashioning plan, which state that the speaker's goal is that it be mutually believed that the refashioned plan can **replace** the original plan. This reasoning process is discussed in the next chapter.

4.6.1 An Example

Consider the scenario where the system knows only of a turquoise bird (**bird1**), a black bird (**bird2**), and a white bird (**bird3**) and it has constructed the referring

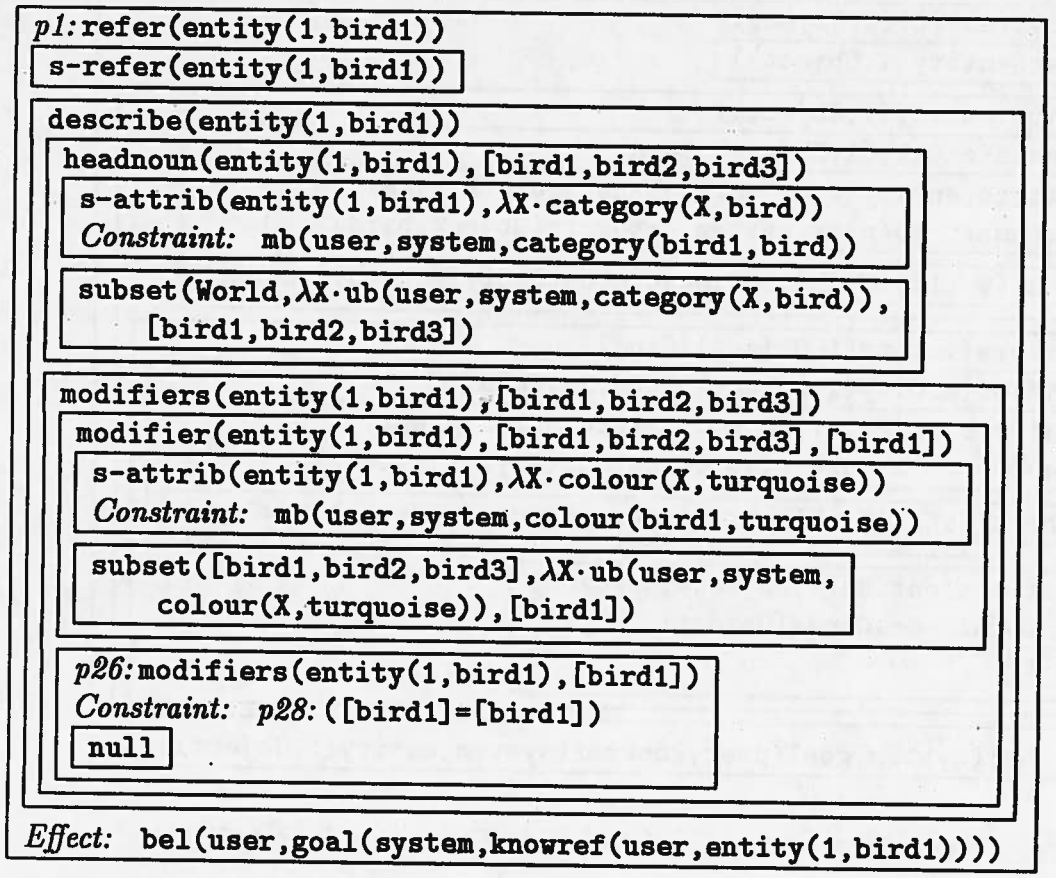


Figure 4.15: Original Referring Expression Plan (p1)

expression plan p1 given in figure 4.15 to refer to the turquoise bird. Let's assume that the system has the following belief:

```
plan(user,p1,knowref(user,entity(1,bird1)))
```

Also, let's assume that the user, after hearing the referring expression, made an utterance, a tentatively voiced *okay*, that was parsed into the following primitive action:

```
s-postpone(p1)
```

The plan inference process takes as input the primitive action, and it derives an instance of **postpone-plan.**[16] The recognized plan is then passed to the plan evaluator. The plan evaluator uses meta-level knowledge to determine the order in which it evaluates the constraints of the plan. Listed below is an edited trace of

[16]The plan recognizer doesn't allow the plan consisting solely of the primitive action **s-postpone**, since it is not a specially marked **end** plan.

31: `postpone-plan(p1)`
Constraint: `error(p1,ErrorNode)`
`plan(system,p1,Goal)`
`constraint(p1,ParentNode,ErrorNode)`
`content(p1,ParentNode,ParentContent)`
`yield(p1,ParentNode,[])`
`ParentContent=modifiers(Entity,Cand)`
Instantiation: *Goal = knowref(user,entity(1,bird1))*
ParentNode = p26
ErrorNode = p28
ParentContent = modifiers(entity(1,bird1),[bird1])
Entity = entity(1,bird1)
Cand = [bird1]
`s-postpone(Plan)`
Effect: `bel(system,goal(user,mb(user,system,error(p1,ErrorNode))))`

Figure 4.16: Inferred Judgement Plan (p31)

its output that shows both the order in which it evaluated the constraints and the instantiations that were made:

`error(p1,ErrorNode)`

`plan(system,p1,Goal)`
Goal = knowref(user,entity(1,bird1))

`yield(p1,ParentPlan,[])`
ParentPlan = p26

`constraint(p1,p26,ErrorNode)`
ErrorNode = p28

`content(p1,p26,ParentContent)`
ParentContent = modifiers(entity(1,bird1),[bird1])

`ParentContent = modifiers(Entity,Cand)`
Entity = entity(1,bird1)
Cand = [bird1]

The evaluated plan is shown in figure 4.16, and the effect of the inferred plan is the following:

`bel(system,goal(user,mb(user,system,error(p1,p28))))`

4.7 Summary

We use discourse plans to account for the conversational moves of Clark and Wilkes-Gibbs (1986). These discourse plans reason about and manipulate the referring expression plans that they take as a parameter. A key idea in our approach is that we separate the discourse plan from the underlying plan. This is evidenced through the surface speech acts that we proposed, which take as a parameter the surface speech acts of the underlying plan. Also, it is evidenced in the way discourse plans are evaluated, in which the successful evaluation of the discourse plan does not depend on the success of the underlying plan.

The discourse plans that we presented are not meant to be a complete set that can account for every type of referring expression clarification. Instead, they are intended to illustrate our approach of using the planning paradigm to account for the generation and understanding of clarifications, and to other clarification subdialogues as well.

One major drawback concerning the plans that we propose is that they are unable to do sophisticated reasoning about an invalid plan derivation to determine why it failed. Instead, we have relied on reasoning about structural and syntactic features of a plan derivation in order to determine which constraint or mental action is the cause of the error and whether the referring expression overconstrains or underconstrains the choice of referent.

Chapter 5

Modeling the Subdialogue

In chapter 3 and 4, we showed how initial referring expressions, and judgement and refashionings of them, can be constructed and inferred in our model. In this chapter, we account for how these tasks fit into a complete model of how an agent collaborates in making a referring action successful. To do this, we need to address two issues. First, the initial referring expression might not be accepted. Hence we need to account for how the discourse plans that we proposed for judging and refashioning the initial referring expression can account for subsequent conversational moves that are made in the subdialogue. We start with the work of Clark and Wilkes-Gibbs (1986) in modeling referring as a collaborative process, and augment their model with the work of Clark and Schaefer (1989) on how conversational participants contribute to discourse.

Second, we have assumed that when constructing a judgement or refashioning plan, an agent is trying to achieve a goal. So, we need to account for how these goals arise from the previous utterance. Previous natural language systems that use plans to account for the surface speech acts underlying an utterance (Cohen and Perrault, 1979; Allen and Perrault, 1980; Appelt, 1985a; Litman and Allen, 1987) model only the recognition or only the construction of an agent's plans, and so do not address this issue. We propose that agents adopt goals to judge and refashion referring expressions because they are mutually responsible for the success of the referring action. We then present our model that incorporates this mutual responsibility and that can account for how judgement and refashioning goals arise.

To conclude the chapter, we present an example of the processing of our model. The example illustrates how discourse goals arise when collaborating to make a referring expression and how these goals can be achieved through the judgement and refashioning plans that we presented for initial referring expressions.

5.1 Subsequent Judgements and Refashionings

In the previous chapter, we presented plans for judging and refashioning an initial referring expression. The aim of this section is to show that these plans can also account for the subsequent conversational moves that participants perform in making a referring action successful. To do this, we first examine the model that Clark and Wilkes-Gibbs (1986) propose. However, their model has an implicit assumption that judgements and refashionings are always accepted. So, we augment their model by adopting the proposal of Clark and Schaefer (1989) that judgements and refashionings are also subjected to an acceptance process. The resulting model can then account for how judgements and refashionings contribute to discourse in a consistent manner with how referring expressions contribute to discourse. Thus we can better justify our usage of the same discourse plans to account for subsequent judgements and refashionings as are used for judging and refashioning the initial referring expression.

5.1.1 Acceptance Processes

Clark and Wilkes-Gibbs (1986) propose that conversational participants employ an *acceptance process* when collaborating to make a referring action successful, and that this process accounts for the conversational moves that participants make:

> The acceptance process is played out in conversation ... as a series of steps. ... The basic process, which might be called the *acceptance cycle*, consists of a presentation plus its verdict. Let x, y, and z stand for noun phrases or their emendations. A presents x and then B evaluates it. If the verdict is not positive, then A or B must refashion that presentation. That person can offer: a repair x', an expansion y, or a replacement z. The refashioned presentation, whether x', $x + y$, or z, is evaluated, and so on. Acceptance cycles apply iteratively, with one repair, expansion, or replacement after another, until a noun phrase is mutually accepted. With that, A and B take the process to be complete. (p. 24)

Although not the theme of their work, Clark and Wilkes-Gibbs assume that the acceptance process is employed in order to *contribute* the referring expression to the discourse. The result of the contribution is that the final referring expression is added to the *common ground*[1] of the participants, and hence provides a basis for

[1] We are using *common ground* following Clark and Schaefer's work (1989) on contributing to discourse. Like them, we do not attempt to formally define this term. It is, however, usually taken as what is mutually believed by the conversational participants, or some weaker variant (Clark and Marshall, 1981; Perrault and Cohen, 1981).

future references to the object being referred to, and helps to establish a perspective for describing other objects.

However, in their account of the acceptance process, Clark and Wilkes-Gibbs fail to explicitly mention an important key to understanding this process. So, we draw on the work of Clark and Schaefer (1989), in which they point out that "each part of the acceptance phase is itself a contribution" (p. 269). So, there is an acceptance process for each judgement and refashioning of the referring expression, and once the judgement or refashioning is accepted, the common ground of the participants is updated. Thus after a refashioning has been uttered and accepted, the refashioned referring expression will be added to the common ground, and thus available for the other participant to judge it. Likewise, after a judgement has been uttered and accepted, it is added to the common ground and can be used as the basis of the next refashioning.

From the preceding discussion, it should be clear that the judgement and refashioning moves are subjected to an acceptance process that is separate from the acceptance process of the referring action. So, we can distinguish between 1) accepting a refashioning move and 2) accepting the result of the refashioning—the refashioned referring expression. In fact, Clark and Wilkes-Gibbs implicitly assume that a refashioning move is always accepted, even if the refashioned referring expression that it proposes is not accepted. So, the acceptance of the refashioning move need not depend on the refashioned referring expression. This implies that there is a difference between 1) rejecting a refashioning move and 2) accepting the refashioning move and rejecting the refashioned referring expression. But is this difference significant to a computational model that accounts for how an agent collaborates in making a referring expression? We feel that it is, for it impacts both the determination of the common ground that is being built up, and it impacts the discourse structure.

Consider dialogue (5.1), which was collected at an information booth in a Toronto train station (Horrigan, 1977). (Although the participants are not collaborating in making a referring expression, the dialogue will serve to illustrate our point.)

(5.1) **P:** [1] The 8:50 to Montreal?
C: [2] 8:50 to Montreal. Gate 7.
P: [3] Where is it?
C: [4] Down this way to your left. Second one on the left.
P: [5] OK. Thank you.

Litman (1985), in her work on recognizing clarification subdialogues gives the following analysis of how her model, as the clerk, interprets the third utterance.

> The first preference [that is, assuming that the clerk's response was understood] fails. ...The second preference succeeds, and the utterance is recognized as part of an introduction of a new identify-parameter referring to the old one. (p. 83)

In other words, the utterance "Where is it?" is interpreted as rejecting the contribution of "Gate 7" and initiating a *side sequence* to accept it. From this analysis, Litman claims that a coherent continuation of the dialogue would be for the passenger to utter "What's a gate?" as the fifth utterance, because it would contribute towards the acceptance of "Gate 7."

However, we feel that Litman's analysis of the dialogue is incorrect. Our intuition leads us to believe that the third utterance, "Where is it?", actually accepts the clerk's contribution of "Gate 7", and hence does not initiate a side sequence to accept it. So, if the passenger utters "What's a gate?" as the fifth utterance, this utterance cannot be interpreted as a contribution towards the acceptance of "Gate 7" since it was already accepted.[2] In examining Litman's analysis, it is interesting to note that her model does first try to to interpret the third utterance as accepting the previous contribution. However, this fails because the "SURFACE-REQUEST does not match (directly or by chaining) any of the steps of PLAN3" (1985, p. 83), where PLAN3 is the plan that the second utterance is contributing towards. We feel that it does not match any of the steps of PLAN3 because PLAN3 is not updated to reflect the additional information that the train departs from gate 7.[3]

As the above discussion shows, if a model fails to properly determine whether a contribution has been accepted or not, it will make undesirable expectations of how a discourse will continue, and it will fail to properly account for how the common ground is being updated during a dialogue.

5.1.2 Judging Contributions

As we have already mentioned, the acceptance of refashioning moves should not depend on the judgement of the refashioned referring expression. So, what should it

[2]We view "What's a gate?" as a semantic return (see Allen 1987).

[3]This brings up the related issue of when a speaker should update the common ground to reflect her contribution. This is an important question, since to understand the next utterance the speaker might have to presuppose the acceptance of her utterance, even though she might have no evidence for doing so.

depend on? Clark and Schaefer (1989) propose that the acceptance of a contribution depends on whether the hearer "believes he is understanding well enough for current purposes" (p. 267). However, they do not define *current purposes.*

By viewing language as goal-oriented behaviour, we can differentiate the purpose of a contribution from the purpose of what it is contributing towards. For judgement moves, it is to inform the hearer of the speaker's judgement of the referring expression, and for refashioning moves, it is to replace the current referring expression by a refashioned referring expression. So, we propose that the hearer uses the following criteria to decide if he has understood a judgement or refashioning move. First, the hearer must identify that the speaker is making a judgement or refashioning move. Second, he must identify the relationship that holds between these moves and the current discourse structure. Third, for a judgement move, he must determine why the speaker finds the current referring expression invalid (in our model, this will be to identify the constraint or mental action that was violated), and for refashioning moves, he must determine the referring expression that the speaker intends to replace the current referring expression by.[4]

A point that we would like to address is why the acceptance of a refashioning move should not depend on the acceptance of the refashioned referring expression. We feel that if a hearer has understood a refashioning but does not find the refashioned referring expression acceptable, he has a choice between 1) contributing towards the refashioning move, or 2) accepting the refashioning move and contributing towards the referring action. However, by not accepting the refashioning, the participants must suspend the current dialogue while they collaborate on the refashioning. Plus, assuming that participants presuppose the acceptance of their contributions, by breaking this expectation, the rejection of the refashioning must be explicitly marked in order for it to be properly interpreted. Hence, we conjecture that there is an advantage, in terms of minimizing collaborative effort, in accepting the refashioning and contributing to the refashioned referring expression.

5.1.3 Our Model of Judging Contributions

In this section, we show how the criteria for understanding a judgement or refashioning move can be incorporated into our model. Since we model judgement and refashioning moves as plans, we formulate these criteria in terms of plan recognition and plan

[4]This is obviously a simplification. A hearer might meet all of these criteria and still not accept the contribution. For instance he might want to question the other person's beliefs, "Why do you think it's green?" Or he might not meet all of the criteria. For instance he might make an assumption in order to meet the criteria, and rely on the speaker to correct him if his assumption is wrong.

evaluation.

From the plan recognition process, the hearer might have several candidate plan derivations that can account for the speaker's utterance, especially in the case of refashioning plans where the same surface speech act is used for both expansions and replacements. Each of the candidate plans is evaluated, as is explained below. If there is exactly one valid plan derivation, then the hearer will believe he has understood the contribution and will accept it. Otherwise, he will believe that he has not understood the utterance well enough and will not accept it, and so he will contribute towards its acceptance, for instance by uttering "What?"[5]

Now we just have to account for how our model evaluates refashioning plans. Recall that this evaluation should not depend on the evaluation of the referring expression that it proposes to replace the current referring expression by. As was outlined in section 4.6, our refashioning plans include the mental actions `construct` and `evaluate`. When the evaluator encounters the `construct` action in a derivation, it performs plan recognition to find a plan that incorporates the primitive actions and that accomplishes the goal; however, it does not evaluate the recognized plan. When the evaluator encounters the `evaluate` action in a derivation, it does not evaluate the plan that is its parameter, since the hearer does not know whether the speaker intends for him to find it valid (for instance, the speaker might think that the plan depends on knowledge not available to the hearer).[6] Hence in evaluating a refashioning plan, the refashioned referring expression plan is not evaluated; however, it is recognized and can thus serve to replace the current referring expression.[7]

5.1.4 Discourse Plans

Now that we have the above framework in place, we can account for how subsequent judgements and refashionings can be accounted for by the same discourse plans as are used for judging and refashioning an initial referring expression. After a refashioning move has proposed a new referring expression and the refashioning move has been accepted, the new referring expression is added to the common ground of the conversational participants, and so is available to be judged and refashioned in the same

[5] In this thesis, we do not give any plan schemas that represent such utterances.

[6] This brings up an interesting question: How can a hearer determine whether the speaker intends him to recognize a plan and whether he should find the recognized plan as being valid?

[7] Another approach would have been to propose that the refashioning plans do not construct the new referring expression, but only inform the hearer of the new plan. So, the new plan would be constructed independently of the refashioning plan. However, with this approach, the hearer would not be able to recognize the new referring expression plan by recognizing the refashioning plan, and hence we would need a way to account for how this would be done. See Lambert and Carberry (1991) and Ramshaw (1991) for a division between discourse plans and problem-solving plans.

was as the initial referring expression is. So, as Clark and Wilkes-Gibbs propose, the same process for judging and refashioning the initial referring expression is used to judge and refashion subsequent refashionings of the referring expression. In terms of our model where plans are used to account for judging and refashioning the initial referring expression, these same plans are used for subsequent refashionings of the referring expression.

5.2 How Discourse Goals Arise

Throughout this thesis, we have been operating under the view that language is goal-oriented behaviour. Since the model that we are proposing accounts for how a conversational participant collaborates in making a referring expression, we must show how a participant adopts goals that lead to the acceptance of a referring expression.

Clark and Wilkes-Gibbs propose that conversational participants are *mutually responsible* for the acceptance of a referring expression. We feel that it is this mutual responsibility that accounts for why a participant contributes to the acceptance of a referring expression, and why he or she will have expectations that the other participant will also be contributing.

The question now arises as to how this mutual responsibility should be represented so that it can be used to justify why a participant adopts goals to contribute to the acceptance of a referring action. To do this, we not only have to represent that participants are mutually responsible for achieving some goal, but also that they intend to achieve the goal,[8] and that they might already have contributed towards a plan in attempting to do so. We represent this by the predicate `mgoal`, which takes as its parameters the participants involved in the collaboration, the plan that they are currently considering, and the goal.

mgoal(Agt1,Agt2,Plan,Goal): Agents `Agt1` and `Agt2` intend to achieve `Goal` and are currently considering `Plan` as a plan to achieve this goal.

By *currently considering*, we mean that the agents have a set of beliefs (or perhaps more accurately, mutual beliefs) about what the current plan is composed of, and a belief that the current plan is *coherent*.[9] For expository reasons, we refer to the plan

[8] We are purposely avoiding the issue of whether the participants have a mutual intention. See (Searle, 1990) for a discussion of this issue.

[9] Our set of beliefs about the current plan is similar to Pollack's *EPLAN* (1990, p.93). However, since the actions have already been performed, we do not have to consider if the actions of the plan are executable or if some agent intends to perform the actions. Note that the current plan need not be valid.

that is the parameter of an `mgoal` as a *mutual plan.*

Now that we have proposed that the collaboration results from an `mgoal`, we need to account for how an agent adopts an `mgoal`. We also need to account for how goals to contribute to the acceptance of a mutual plan arise, and the effect of an utterance on the speaker's and hearer's beliefs. We express this knowledge as a set of rules that specify the conditions under which an `mgoal`, belief, or goal can be adopted.[10] These rules shouldn't be seen as the definitive answer, but only as a starting point towards a theory that can account for how agents collaborate.

One simplification that we have made in the rules is that we assume that the judgement and refashioning plans are always valid. This has been done because our thesis is about how agents collaborate on making a referring action successful, and not about repairs in general. However, since we have incorporated Clark and Schaefer's work on how agents contribute to discourse, we feel that our model can be extended to handle such situations.

5.2.1 Understanding an Utterance

To understand an utterance, it is necessary to determine the goal that the speaker intends to achieve by her utterance. We start with the observed primitive actions that were derived by the parser, and use plan recognition to derive the derivation. We then evaluate the constraints and mental actions of the plan to determine whether the plan is valid. Next, the effects of the plan instances in the plan derivation are analyzed. These effects are what the hearer believes given that he has inferred the plan. In particular, the effect of the root plan in the derivation is of the form `bel(Hearer,goal(Speaker,Goal))`, where `Goal` is the goal that the speaker intends to achieve by her plan.[11]

From this process, the hearer updates his beliefs to capture the information that was inferred. This allows the rules to easily make use of the information. First, a belief is added that the speaker has the inferred goal, and is using the inferred plan as a means of accomplishing this goal:[12]

```
plan(Speaker,Plan,Goal)
```

[10] In stating these rules, we use the same operators to express adopting an `mgoal`, belief, or goal. For a more formal account, three different operators should be used.

[11] The effects are formulated in this way because we are assuming that when a speaker has a communicative goal, she plans to achieve the goal by making the hearer recognize it.

[12] We do not add the belief that the speaker believes her plan will achieve the goal, since the speaker might realize her plan is invalid, and choose to communicate this plan to the hearer with the intention that the two participants will collaborate in order to achieve the goal.

Second, a belief is added to indicate which agent contributed the plan:

```
contributed(Speaker,Plan)
```

Third, a belief is added to indicate whether the plan is valid. If it is valid, then `Result` is instantiated to `success`; otherwise, `Result` is instantiated to `failure(Node)`, where `Node` is the node that the error occurred at.

```
evaluation(Plan,Result)
```

5.2.2 Adopting Discourse Goals

The hearer's beliefs about the speaker's utterance are used to drive a chain of processing that culminates in the adopting of discourse goals to either judge or refashion the referring expression.

Adopting an `mgoal`

The first rule that we give is for adopting an `mgoal`. As with `goal`, we assume that once a participant has adopted an `mgoal`, he believes that he has adopted it. By adopting an `mgoal`, the hearer intends to adopt discourse goals to collaborate in order to achieve the goal of the `mgoal`. Note, however, that not all goals that are inferred are necessarily intended to become the basis of an `mgoal`, nor will a participant necessarily adopt an `mgoal` even if the speaker intended him to. So, this rule needs to be qualified. In this thesis, we have taken the approach of adopting an `mgoal` for any plan that the speaker has (which the hearer knows about) in which the goal is `knowref`.[13]

Rule 1[14]

```
mgoal(Hearer,Speaker,Plan,Goal) ⇐
        plan(Speaker,Plan,Goal) &
        Goal = knowref(Hearer,Entity)
```

Adopting Beliefs

We give five rules for adopting beliefs about an `mgoal`.

[13]The value of the discourse entity `Entity` is not relevant for this rule.

[14]The rules also include the predicates `speaker(Speaker)` and `hearer(Hearer)` to instantiate the variables `Speaker` and `Hearer`.

Rule 2 is used to adopt the belief that the speaker believes there is an error in Plan, given that Plan is a mutual plan and that the speaker has another plan, SPlan, to make it mutually believed that there is an error in Plan.

Rule 2

```
bel(Speaker,error(Plan,ErrorNode)) ⇐
        mgoal(Hearer,Speaker,Plan,Goal) &
        plan(Speaker,SPlan,mb(Hearer,Speaker,error(Plan,Node)))
```

Rule 3 is used to adopt the belief that there is an error in Plan, given that Plan is a mutual plan and that the speaker has a plan, SPlan, to make it mutually believed that there is an error in Plan. This rule has two built-in assumptions. The first, as has already been mentioned, is that judgement plans are always valid; in other words, evaluation(SPlan,success) is true. The second assumption is that the hearer accepts the judgement. Since both participants are collaborating on the plan, both agents must find the plan to be valid, and so the hearer can either convince the speaker that there is not in fact an error, or accept the speaker's belief even if he doesn't himself think there is such an error. We pursue only the latter option.

Rule 3

```
error(Plan,Node) ⇐
        mgoal(Hearer,Speaker,Plan,Goal) &
        plan(Speaker,SPlan,mb(Hearer,Speaker,error(Plan,Node)))
```

Rule 4 is also used to adopt a belief given that the speaker has this as the goal of her plan, SPlan. In this case, the speaker's goal is that it to be mutually believed that a new plan replaces the current mutual plan. As a result of adopting this belief, the hearer (through the belief module) will update mgoal by replacing Plan with NewPlan, and will evaluate NewPlan to determine whether it is valid. There are two built-in assumptions in this rule. The first is that refashioning plans are always valid. The second, which is the subject of section 5.1.2, is that the hearer always accepts valid refashioning plans.

Rule 4

```
replace(Plan,NewPlan) ⇐
        mgoal(Hearer,Speaker,Plan,Goal) &
        plan(Speaker,SPlan,mb(Hearer,Speaker,replace(Plan,NewPlan)))
```

Rule 5 is used to adopt the belief that Plan achieves its goal. The conditions specify that Plan is a mutual plan, that it is valid, and that the goal, which is a proposition, is true.

Rule 5

```
achieve(Plan,Goal) ⇐
        mgoal(Hearer,Speaker,Plan,Goal) &
        evaluation(Plan,success) &
        true(Goal)
```

Rule 6 is used to adopt the belief that a mutual plan has an error in it, and so does not achieve its goal.

Rule 6

```
error(Plan,Node) ⇐
        mgoal(Hearer,Speaker,Plan,Goal) &
        evaluation(Plan,failure(Node))
```

Adopting Goals

The next set of rules capture how an agent adopts goals in order to collaborate in achieving the goal that is the parameter of an mgoal. We refer to the agent who is adopting a goal as the speaker, and so the conditions under which the rules apply are stated with respect to the speaker's point of view.

Rule 7 is used to adopt the goal of informing the hearer that there is an error in Plan. The conditions specify that Plan is a mutual plan, that there is an error in the plan, and that neither agent already has a plan, APlan, to make it mutually believed that there is an error in Plan.

Rule 7

```
goal(Speaker,mb(Speaker,Hearer,error(Plan,Node))) ⇐
        mgoal(Speaker,Hearer,Plan,Goal) &
        error(Plan,Node) &
        not(plan(Agt,APlan,mb(Speaker,Hearer,error(Plan,Node))))
```

Rule 8 is used to adopt the goal of replacing a plan, Plan, that has an error. The conditions for applying this rule specify that Plan is a mutual plan, that both the speaker and hearer believe there is an error in Plan, and that neither agent already has a plan, APlan, to replace it.[15]

[15]The statement of this goal is problematic. It really states that the hearer has a goal of finding a new plan to replace the old plan, and communicating this new plan to the speaker.

Rule 8

```
goal(Speaker,mb(Speaker,Hearer,replace(Plan,NewPlan))) ⇐
      mgoal(Speaker,Hearer,Plan,Goal) &
      error(Plan,Node) &
      bel(Hearer,error(Plan,Node)) &
      not(plan(Agt,APlan,mb(Speaker,Hearer,replace(Plan,NewPlan))))
```

Rule 9 is used to adopt the goal of communicating the speaker's acceptance of the mutual plan. This rule ensures that the agent who proposed the plan doesn't initiate the acceptance of it.

Rule 9

```
goal(Speaker,mb(Speaker,Hearer,achieve(Plan,Goal))) ⇐
      mgoal(Speaker,Hearer,Plan,Goal) &
      achieve(Plan,Goal) &
      contributed(Hearer,Plan)
```

5.2.3 Achieving Discourse Goals

Now that we have seen how the speaker adopts goals, we can focus on how the speaker can construct a plan to achieve these goals. First, the goals that we are interested in involve collaborating with another agent, and involve the other agent either adopting a belief, updating the mutual plan, or identifying an object. Since these goals cannot be directly achieved by a plan of action, the speaker must instead plan actions that will indirectly achieve them, for instance by planning an utterance that results in the hearer recognizing her goal. So, if the speaker has the goal `Goal`, she will attempt to construct a plan whose effect is `bel(Hearer,goal(Speaker,Goal))`. After such a plan is constructed, this fact is added to the speaker's beliefs, along with the fact that the speaker contributed it, and that it is a valid plan.

```
plan(Speaker,Plan,Goal)
contributed(Speaker,Plan)
evaluation(Plan,success)
```

The speaker also adopts `mgoals` and beliefs to reflect the effect of the constructed plan. The rules that we give to account for this process are similar to the rules given in the previous section for adding an `mgoal` or a belief on the basis of an inferred plan. However, these rules are from the point of view of the speaker rather than the hearer.

Rule 10 is a duplicate of rule 1, and is used by the speaker to adopt an mgoal in anticipation of the hearer adopting it.

Rule 10

```
mgoal(Speaker,Hearer,Plan,Goal) ⇐
      plan(Speaker,Plan,Goal) &
      Goal = knowref(Hearer,Entity)
```

Rule 11 is a duplicate of rule 4 and is used to update the mgoal. This rule presupposes the hearer's acceptance of the speaker's plan, SPlan. Note that it is this rule, and not an action in the constructed plan, that actually replaces the old referring expression plan by a new plan.

Rule 11

```
replace(Plan,NewPlan) ⇐
      mgoal(Speaker,Hearer,Plan,Goal) &
      plan(Speaker,SPlan,mb(Speaker,Hearer,replace(Plan,NewPlan)))
```

Rule 12 is used to adopt the belief that the hearer believes there is a error in the mutual plan.[16]

Rule 12

```
bel(Hearer,error(Plan,Node)) ⇐
      plan(Speaker,SPlan,mb(Speaker,Hearer,error(Plan,Node)))
```

Rule 13 is a duplicate of rule 5.

Rule 13

```
achieve(Plan,Goal) ⇐
      mgoal(Speaker,Hearer,Plan,Goal) &
      evaluation(Plan,success) &
      true(Goal)
```

[16]Its counterpart is rule 2, but due to the inconsistent way that mutual belief is handled, it is not identical.

5.3 An Example

In this section, we illustrate how our model accounts for how an agent adopts discourse goals to judge and refashion the current referring expression. We focus on the application of the rules that we presented in the previous section and the beliefs that are adopted about the plan that is constructed or inferred. In this example, we use dialogue (5.3) and show how the system, taking the role of person A (the initiator), reasons about the plan that it infers from the second utterance, and how this reasoning process leads it to adopting discourse goals that it attempts to achieve in the third utterance.

(5.3) **A:** [1] See the black bird.
B: [2] The small one?
A: [3] No, the large one.
B: [4] Okay.

We first need to provide the appropriate context, in order for the system to understand the second utterance. So, let's assume that the system has the following `mgoal`, where `p1` is the plan derivation corresponding to the referring expression "the black bird."

```
mgoal(system,user,p1,knowref(user,entity(1,bird2)))
```

For the second utterance, the system is the hearer and is given as input the surface speech acts that underlie the utterance "the small one?"

```
s-postpone(Plan)
s-actions(Plan,[s-attrib(entity(1,Object),λX·size(X,small))])
```

Although the two actions `s-postpone` and `s-actions` arise from the same utterance, we assume that the parser has separated them.[17]

Starting with the `s-postpone` action, the system performs plan recognition and infers that the user's plan, `p31`, is an instance of `postpone-plan`. The constraints and mental actions of the plan are then evaluated, and the plan is found to be valid. From the effect of the plan, the system infers that the user's goal is to inform the system that the user finds the referring expression plan underconstrained (`p28` is the constraint of the plan instance of the referring expression plan that terminates the addition of modifiers). From this process, the system adds the following beliefs:

[17] This allows us to consider conversational moves separately, which simplifies our implementation.

```
plan(user,p31,mb(system,user,error(p1,p28))
contributed(user,p31)
evaluation(p31,success)
```

Since plan p1 is a mutual plan, rule 2 and 3 are applied, resulting in the following beliefs:

```
bel(user,error(p1,p28))
error(p1,p28)
```

Next, starting with the s-actions action, the system again performs plan recognition and infers that the user's plan is either an instance of expand-plan or an instance of replace-plan. From evaluating the constraints and mental actions of the two plan derivations, the system finds expand-plan valid, but finds a violation with replace-plan. So, the system chooses expand-plan as the plan underlying the utterance. This leads to the system adopting the following beliefs, where p57 is the referring expression plan that can be glossed as "the small black bird" and p42 is the expand-plan derivation.

```
plan(system,p42,mb(system,user,replace(p1,p57)))[18]
contributed(user,p42)
evaluation(p42,success)
```

The system then applies rule 4, resulting in the following belief being adopted:

```
replace(p1,p57)
```

This causes the belief module to update the mgoal, so that it refers to p57 rather than p1. The belief module also evaluates p57, and it finds that a constraint, p90, on the surface speech act that describes the object's size is unsatisfiable. This results in the following beliefs being adopted:

```
contributed(user,p57)
evaluation(p57,failure(p90))
mgoal(system,user,p57,knowref(user,entity(1,bird2)))
```

[18]Actually, the second parameter of replace is not p57, but the plan derivation corresponding to it.

Rule 6 can now be applied, leading to the following belief:

```
error(p57,p90)
```

This exhausts the beliefs that can be adopted from the user's inferred plans and goals, and so the system now switches from being the hearer of the second utterance to being the speaker of the third utterance. So, the system now checks to see if there are any goals that it can adopt. Using rule 7, it adopts the goal to inform the user that there is an error in the refashioned plan.

```
goal(system,mb(system,user,error(p57,p90)))
```

The system then constructs a plan to satisfy this goal (an instance of `reject-plan`). After constructing this plan, p97, the system adds the following beliefs:

```
plan(system,p97,mb(system,user,error(p57,p90)))
contributed(system,p97)
evaluation(p97,success)
```

This then allows the system to apply rule 12, and so adopt the belief that the user believes there is an error in the referring expression caused by the constraint of the surface speech act that describes the size of the object.

```
bel(user,error(p57,p90))
```

This is the only belief that can be added from the constructed plan, and so the system looks for more goals to pursue. Since the system believes that the user will believe that there is an error in the referring expression plan, it uses rule 8 to adopt the goal to replace it by a new plan.

```
goal(system,mb(system,user,replace(p57,NewPlan)))
```

The system constructs a plan that achieves this goal (an instance of `replace-plan`), and adds the following beliefs about the plan, p107, where p123 is the new referring expression plan.

```
plan(system,p107,mb(system,user,replace(p57,p123)))
contributed(system,p107)
evaluation(p107,success)
```

The system can now add the following belief, using rule 11, to replace the referring expression plan by the new plan.

```
replace(p57,p123)
```

By adding this belief, the belief module updates the `mgoal` (replaces `p57` by `p123`), and adds the following beliefs:

```
contributed(system,p123)
evaluation(p123,success)
mgoal(system,user,p123,knowref(user,entity(1,bird2)))
```

Also, the system adds the belief, sanctioned by rule 13, that it believes the refashioned referring expression plan, `p113`, will allow the user to identity the referent.

```
achieve(p123,knowref(user,entity(1,bird2)))
```

5.4 Summary

In this section, we have drawn on Clark and Wilkes-Gibbs's model of how participants collaborate in making a referring expression, and Clark and Schaefer's work on contributing to discourse. This has allowed us to justify how subsequent judgements and refashionings are modeled, and to motivate our account of how an agent adopts goals to make judgement and refashionings until a referring expression is accepted. So, we have been able to incorporate collaborative activity into a goal-oriented approach to language.

However, as this chapter has shown, there are still many unresolved issues, and many simplifying assumptions that must be removed. In particular, our account of how an agent adopts an `mgoal`, belief, or goal must be set into a more formal model, and an `mgoal` must either be defined as a primitive in this model, such as *we-intention*, or be defined in terms of the other operators. Also, there is the issue of when should an agent adopt an `mgoal`, and what sort of goals he should even consider.

Chapter 6

An Example

In this chapter, we give an example of our system in operation. (The full trace can be found in appendix A). For this example, we use dialogue (6.1) and show how the system collaborates in making the referring action successful.

(6.1) **A:** [1] See the weird creature.
B: [2] In the corner?
A: [3] No, on the television.
B: [4] Okay.

Dialogue (6.1) is based on dialogue (6.2) from the Lund corpus (Svartvik and Quirk, 1980, S.2.4a:1–8).

(6.2) **B:** [1] what's that weird creature over there
C: [2] in the corner
B: [3] mhm
C: [4] it's just a fern plant
B: [5] no the one to the left of it
C: [6] that's [the] television aerial

We do not use the original dialogue, (6.2), because it involves a number of complexities that we haven't addressed in this thesis. First, in the first three lines, the participants are collaborating on a referring expression that is embedded inside a request. The fourth utterance is a response to the request. But, the response shows that the responder did not identify the right referent. So, in the fifth utterance, the

initiator refashions the previous referring expression, which then allows the responder to properly respond to the request.

A second complexity of the original dialogue is that the refashioning in the fifth utterance refers to the referent of the rejected referring expression. So, to process this utterance, our system would need a model of the attentional space and would need to be able to construct and resolve anaphoric references.

A third complexity is that in the first utterance there is a referring expression, "over there", embedded inside of another referring expression. Hence, the second utterance is ambiguous as to which referring expression it is clarifying.

Due to these complexities, dialogue (6.2) could be regarded as a *holy-grail* or benchmark for an eventual perfect system. Since our model and implementation is not at this stage, we have simplified the dialogue so as to remove these complexities, with the result being dialogue (6.1).

We are now ready to show our system in operation. In dialogue (6.1), the system will take the role of person B, who is the responder. In order to account for B's utterances, the system is given the following beliefs:

```
category(fern1,creature)
assessment(fern1,weird)
in(fern1,corner1)
category(antenna1,creature)
assessment(antenna1,weird)
on(antenna1,television1)
category(corner1,corner)
category(television1,television)
world([fern1,antenna1,corner1,television1])
```

So, the system believes that there are two objects that are "weird creatures," a television antenna that is on the television; and a fern plant that is in the corner.

6.1 Understanding "The weird creature"

For the first sentence, the system is given as input the surface speech actions underlying "the weird creature."

```
s-refer(entity(1,Object))
s-attrib(entity(1,Object),λX·assessment(X,weird))
s-attrib(entity(1,Object),λX·category(X,creature))
```

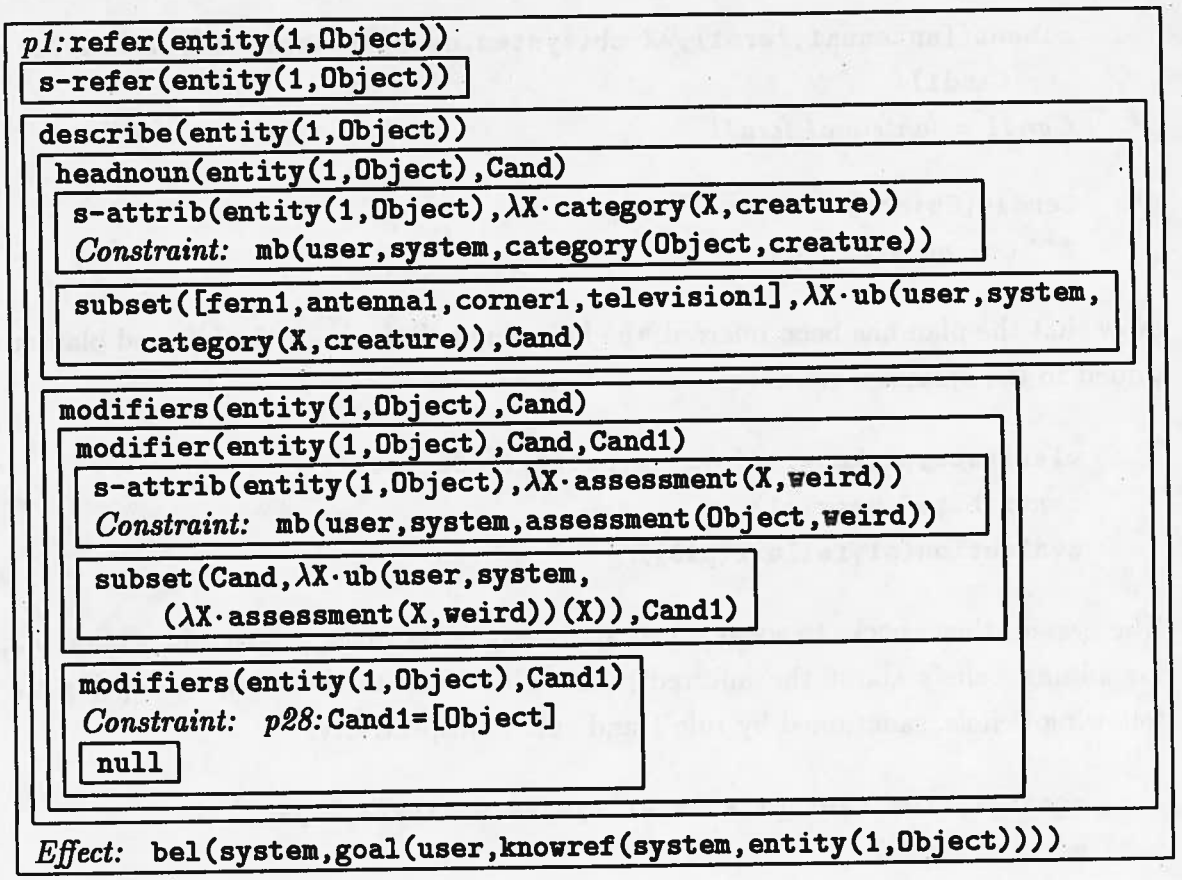


Figure 6.1: Recognized Referring Expression Plan (p1)

The system invokes the plan recognizer to derive a plan derivation whose yield is the set of observed surface speech actions. The resulting plan derivation, p1, is shown in figure 6.1 (with the where-predicates instantiated).

Next, the plan derivation is evaluated. Shown below are the constraints and mental actions that are evaluated, and the variable instantiations that result. As can be seen, the subset action in the headnoun plan is evaluated first, then the subset action in the modifier plan, and then the constraint on the modifiers plan that terminates the addition of modifiers. However, the constraint fails, since the plan evaluator finds that there are two objects that match the description rather than one. In other words, the system cannot determine if the user is referring to the antenna or the fern plant.

subset([fern1,antenna1,corner1,television1],λX·ub(system,user,
category(X,creature)),Cand)
Cand = [antenna1,fern1]

```
subset([antenna1,fern1],λX·ub(system,user,assessment(X,weird)),
    Cand1)
```
Cand1 = [antenna1,fern1]

```
Cand1=[Object]
```
**** cannot be satisfied ****

Now that the plan has been inferred, the following beliefs about the inferred plan are added to the system's beliefs:

```
plan(user,p1,knowref(system,entity(1,Object)))
contributed(user,p1)
evaluation(p1,failure(p28))
```

The system then checks to see if it can apply any of its rules, as given in section 5.2, for adding beliefs about the inferred plan. This leads to the system adopting the following beliefs, sanctioned by rule 1 and rule 6, respectively:

```
mgoal(system,user,p1,knowref(system,entity(1,Object)))
error(p1,p28)
```

6.2 Constructing "In the corner?"

The system next checks if there are any goals that it should adopt. Since it found the referring expression plan invalid, it gives itself two goals. The first goal is to inform the user that the referring expression was not successful and the second is to refashion it. The surface speech actions that are the yield of the plans that it constructs would be realized by "in the corner?"

6.2.1 Judgement Plan

The system gives itself the goal of making it mutually believed that the referring expression plan is invalid. It does this by way of rule 7.

```
goal(system,mb(system,user,error(p1,p28)))
```

The system then constructs a plan to achieve the following effect that is based on this goal:

```
bel(hearer,goal(system,mb(system,user,error(p1,p28))))
```

p31: `postpone-plan(p1)`
Constraint: `error(p1,p28)`
`plan(user,p1,knowref(system,entity(1,Var1)))`
`constraint(p1,ParentPlan,ErrorNode)`
`content(p1,ParentPlan,Content)`
`yield(p1,ParentPlan,[])`
`Content = modifiers(Entity,Cand)`
Instantiation: *Goal = knowref(system,entity(1,Object))*
ParentPlan = p26
Content = modifiers(entity(1,Object),[antenna1,fern1]))
Entity = entity(1,Object)
Cand = [antenna1,fern1]
`s-postpone(p1)`
Effect: `bel(user,goal(system,mb(system,user,error(p1,p28))))`

Figure 6.2: Constructed Judgement Plan (p31)

Since the referring expression is underconstrained, the plan constructor builds an instance of `postpone-plan`, `p31`, as shown in figure 6.2. The yield of this plan is `s-postpone(p1)` and this action would be passed to the generator.[1]

The system then adds the following beliefs about the constructed plan:

```
plan(system,p31,mb(system,user,error(p1,p28)))
contributed(system,p31)
evaluation(p31,success)
```

The system then applies rule 12 and infers that the user will believe that there is an error in the plan.

```
bel(user,error(p1,p28))
```

6.2.2 Refashioning Plan

The system next checks to see if there are any other goals it should adopt. By applying rule 8, it gives itself the goal of refashioning the invalid referring expression plan and of informing the user of the new plan:

```
goal(system,mb(system,user,replace(p1,NewPlan)))
```

[1] We assume that the generator will wait with linguistically realizing this action so as to be able to combine it with the `s-actions` surface speech act that it receives next.

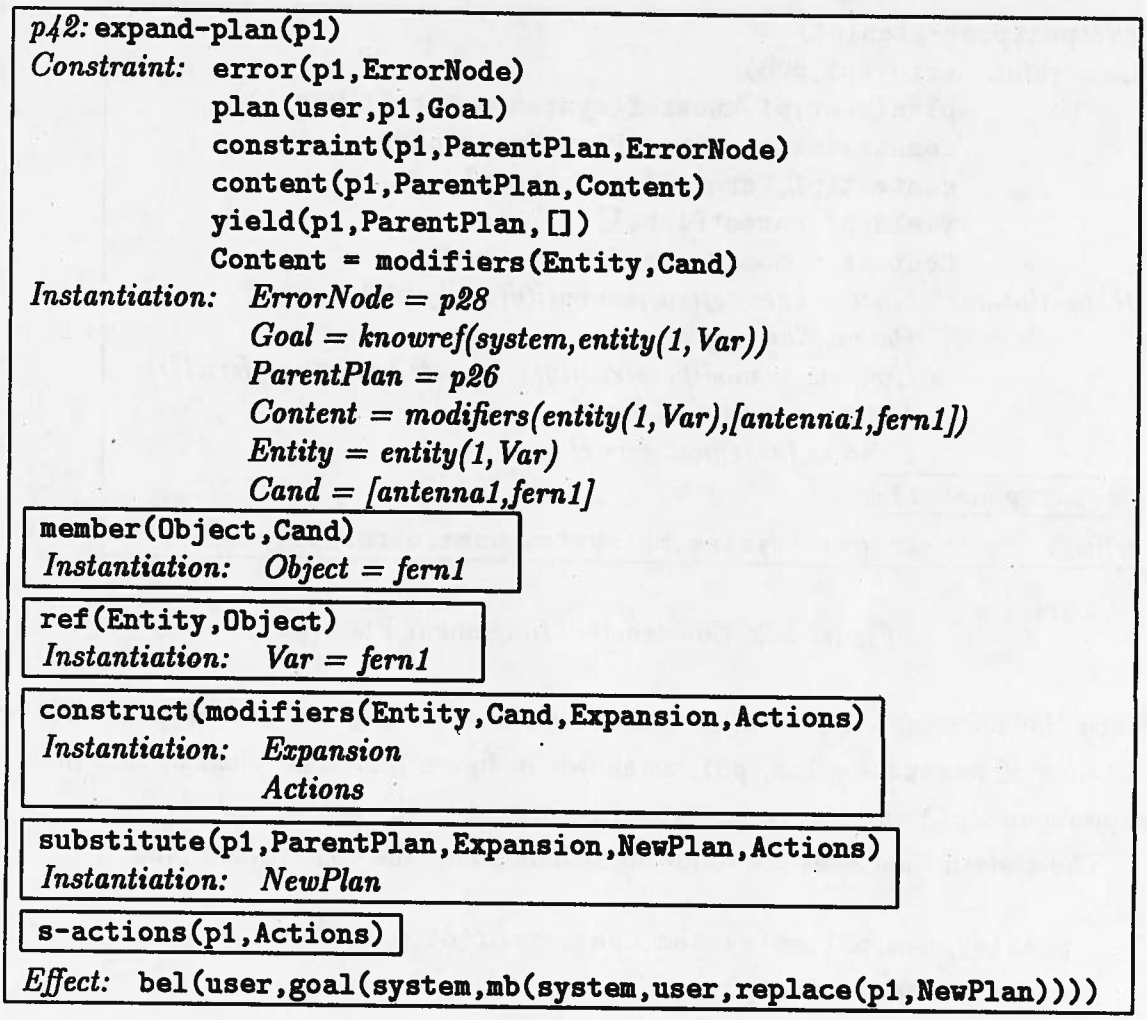


Figure 6.3: Constructed Refashioning Plan (p42)

The system then constructs a plan to achieve the following effect that is based on this goal:

```
bel(user,goal(system,mb(system,user,replace(p1,NewPlan))))
```

The plan constructor builds a plan derivation, **p42**, based on **expand-plan** (figure 6.3)[2]. In constructing this plan, the system chooses one of the objects that matched the original description as the likely referent; in this case it happens to choose the object in the corner. It then constructs an expansion to distinguish this object from the others that matched, and this expansion is incorporated into the old referring expression plan, creating a new expanded plan. The expansion is shown in figure 6.4, and the surface speech actions of the expansion are shown below. These

[2]We do not show the value for the variables **Expansion**, **Actions**, and **NewPlan** due to space and formatting considerations. However, we do indicate where they are instantiated.

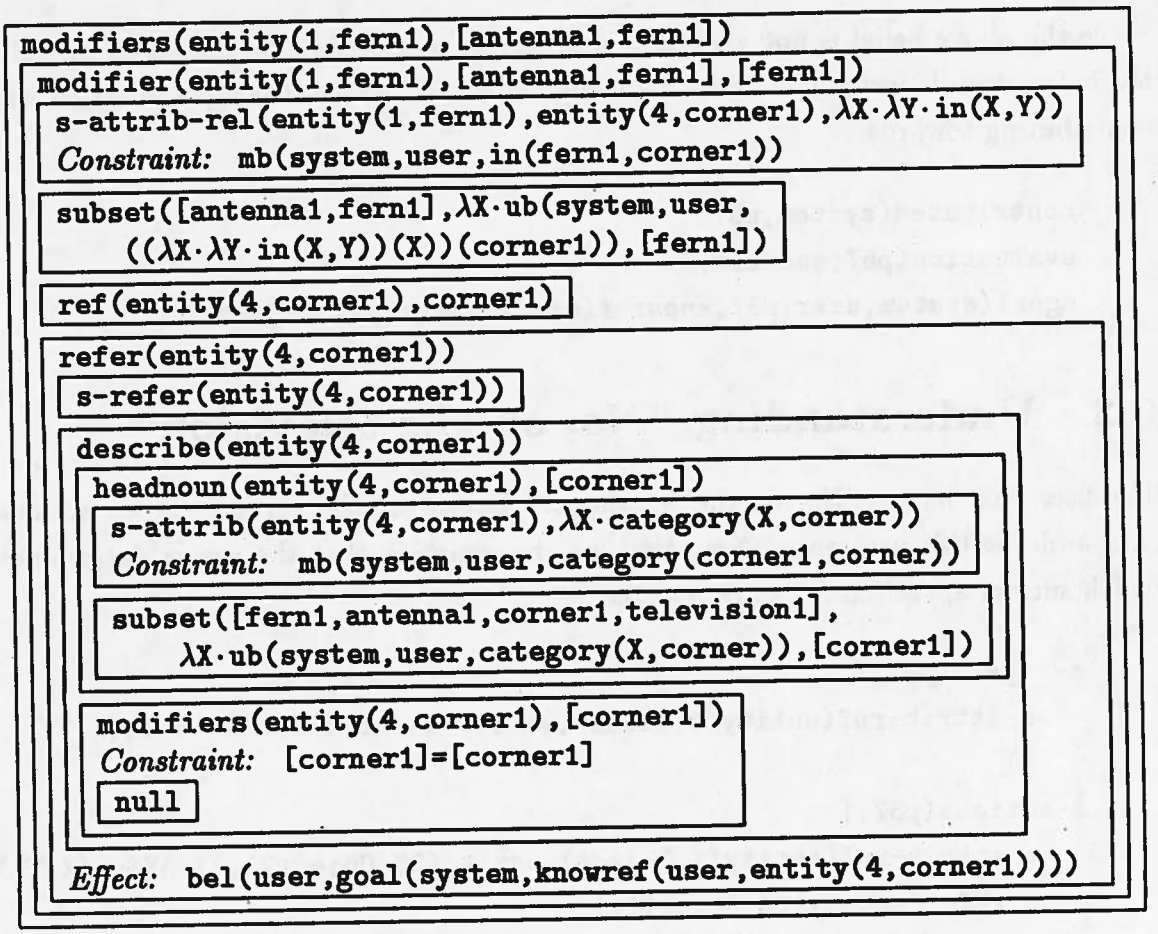


Figure 6.4: Constructed Expansion

actions are embedded inside the surface speech action **s-actions**, which is given to the generator.

```
s-attrib(entity(4,corner1),λX·category(X,corner)),
s-refer(entity(4,corner1)),
s-attrib-rel(entity(1,fern1),entity(4,corner1),λX·λY·in(X,Y))
```

The system then updates its beliefs to reflect the constructed plan:

```
plan(system,p42,mb(system,user,replace(p1,NewPlan)))
contributed(system,p42)
evaluation(p42,success)
```

Next, it applies rule 11 and adds the belief that the new expanded plan, p57, replaces the old referring expression plan, p1.

```
replace(p1,NewPlan)
```

Since the above belief is not viewed as a primitive proposition by the belief module, the belief module instead adds the following beliefs and revises the `mgoal` that `p1` was contributing towards:

```
contributed(system,p57)
evaluation(p57,success)
mgoal(system,user,p57,knowref(system,entity(1,Object)))
```

6.3 Understanding "No, on the television"

The user next utters "No, on the television." Below are the surface speech actions that underlie this utterance. Note that we are assuming that the parser determines which surface speech actions are rejected.[3]

```
s-reject(p57,[
   s-attrib-rel(entity(1,fern1),entity(4,corner1),λX·λY·in(X,Y))
   ])
s-actions(p57,[
   s-attrib-rel(entity(1,Object),entity(38,Object2),λX·λY·on(X,Y)),
   s-refer(entity(38,Object1)),
   s-attrib(entity(38,Object1),λX·category(X,television))
   ])
```

Although the two actions `s-reject` and `s-actions` arise from the same utterance, we assume that the parser has separated them.[4]

6.3.1 Judgement Plan

The system starts with the action `s-reject(p57,Actions)`, where `Actions` is given below:

```
s-attrib-rel(entity(1,fern1),entity(4,corner1),λX·λY·in(X,Y))
```

It invokes the plan recognizer, which derives the derivation, `p119`, shown in figure 6.5 (with the where-predicates instantiated). The system then evaluates the constraints

[3] It could be argued that the surface speech actions that are embedded in `s-reject` should be all three actions that were added by the previous utterance.

[4] This allows us to consider conversational moves separately, which simplifies our implementation.

p119: `reject-plan(p57)`
Constraint: `error(p57,ErrorNode)`
`plan(system,p57,Goal)`
`constraint(p57,ParentPlan,ErrorNode)`
`yield(p57,ParentPlan,Actions)`
`length(Actions,1)`
`s-reject(p57,Actions)`
Effect: `bel(system,goal(user,mb(user,system,error(p57,ErrorNode))))`

Figure 6.5: Recognized Judgement Plan (p119)

and mental actions as shown below:

`error(p57,ErrorNode)`

`plan(system,p57,Goal)`
Goal = knowref(system,entity(1,Object))

`yield(p57,ParentPlan,Actions)`
ParentPlan = p86

`constraint(p57,ParentPlan,ErrorNode)`
ErrorNode = p91

`length(Actions,1)`

Note that the plan evaluator evaluates `yield`, which results in `ParentPlan` being instantiated. This allows `constraint` to be evaluated, resulting in `ErrorNode` being instantiated to `p91`. So, the system is able to determine why the referring expression is not acceptable, which is because the constraint `p91` does not hold.

After inferring the user's plan, the system updates its beliefs:

`plan(user,p119,mb(system,user,error(p57,p91)))`
`contributed(user,p119)`
`evaluation(p119,success)`

The system then applies rule 2 and rule 3, and so adopts the beliefs that the referring expression is overconstrained:

`bel(user,error(p57,p91))`
`error(p57,p91)`

With these beliefs, the system will have the context that it needs to understand the user's refashioning plan.

p129: replace-plan(p57)
Constraint: error(p57,ErrorNode)
plan(system,p57,Goal)
constraint(p57,ParentPlan,ErrorNode)
step(p57,ModifierPlan,ParentPlan)
content(p57,ModifierPlan,Content)
Content = modifier(Entity,Cand,Cand1)
member(Object1,Cand)
ref(Entity,Object1)
construct(modifier(Entity,Cand,Cand1),Expansion,Actions)
substitute(p57,ModifierPlan,Expansion,NewPlan,Actions])
evaluate(NewPlan)
s-actions(p57,Actions)
Effect: bel(system,goal(user,mb(user,system,replace(p57,NewPlan))))

Figure 6.6: Recognized Refashioning Plan (p129)

6.3.2 Refashioning Plan

The system next performs plan recognition starting with the surface speech action s-actions(p57,Actions), where Actions is given below:

s-attrib-rel(entity(1,Object),entity(38,Object2),λX·λY·on(X,Y))
s-refer(entity(38,Object1))
s-attrib(entity(38,Object1),λX·category(X,television))

The resulting derivation, p129, is shown in figure 6.6 (with the where-predicates instantiated). Next it evaluates the constraints and mental actions as shown below:[5]

error(p57,ErrorNode)
ErrorNode = p91

plan(system,p57,Goal)
Goal = knowref(system,entity(1,Object))

constraint(p57,ParentPlan,ErrorNode)
ParentPlan = p86

[5]As in figure 6.3, we do not show the values for Expansion, Actions, and NewPlan. However, we do indicate where they are instantiated.

```
step(p57,ModifierPlan,ParentPlan)
ModifierPlan = p83

content(p57,ModifierPlan,Content)
Content = modifier(entity(1,fern1),[antenna1,fern1],[fern1])

Content = modifier(Entity,Cand,Cand1)
Entity = entity(1,fern1)
Cand = [antenna1,fern1]
Cand1 = [fern1]

ref(Entity,Object1)
Object1 = fern1

member(Object1,Cand)

construct(modifier(Entity,Cand,Cand1),Expansion,Actions)
Expansion

substitute(p57,ModifierPlan,Expansion,NewPlan,Actions)
NewPlan

evaluate(NewPlan)
```

From evaluating `construct`, the variable `Expansion` is instantiated and its value (with the where-predicates instantiated) is shown in figure 6.7. Although the value for `NewPlan` is not shown, it has been instantiated to a plan derivation, `p145`, by the `substitute` action.

Now that the system has inferred the user's plan, `p129`, it updates its beliefs:

```
plan(user,p129,mb(system,user,replace(p57,NewPlan)))
contributed(user,p129)
evaluation(p129,success)
```

The system then applies rule 4 and adopts the belief that the new referring expression plan, `p145`, replaces the old plan, `p57`.

```
replace(p57,NewPlan)
```

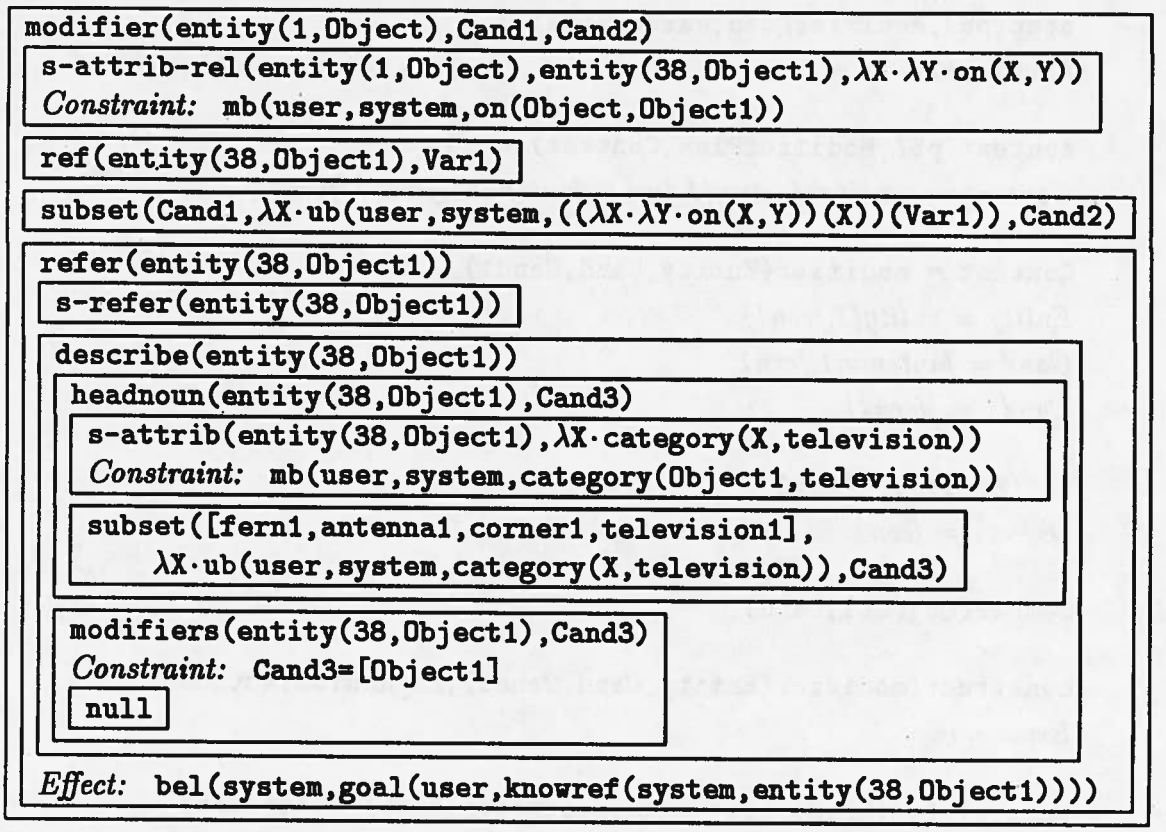


Figure 6.7: Recognized Expansion

Since the belief module does not view this as a primitive proposition, the belief module adds the following beliefs and revises the `mgoal` that `p57` contributed towards:

```
contributed(user,p145)
mgoal(system,user,p145,knowref(system,entity(1,Object)))
```

The belief module also invokes the plan evaluator on the new referring expression plan. Shown below are the constraints and mental actions that it evaluates:

```
subset([fern1,antenna1,corner1,television1],λX·ub(system,user,
    category(X,creature)),Cand)
```
Cand = [antenna1,fern1]

```
subset(Cand,λX·ub(system,user,assessment(X,weird)),Cand1)
```
Cand1 = [antenna1,fern1]

```
ref(entity(38,Object1),Var1)
```
Object1 = Var1[6]

[6]This means that the two variables are now co-referential.

```
subset([fern1,antenna1,corner1,television1],λX·ub(system,user,
    category(X,television)),Cand3)
```
Cand3 = [television1]

```
Cand3=[Object1]
```
Object1 = television1

```
subset(Cand1,λX·ub(system,user,on(X,Var1)),Cand2)
```
Cand2 = [antenna1]

```
mb(system,user,category(Object1,television))
```

```
[antenna1]=[Object]
```
Object = antenna1

```
mb(system,user,category(Object,creature))
```

```
mb(system,user,assessment(Object,weird))
```

```
mb(system,user,on(Object,Object1))
```

Since the plan evaluator finds the referring expression plan p145 valid, the following belief is added:

```
evaluation(p145,success)
```

The system can now apply rule 5. For convenience, we have repeated rule 5 below, with the variables instantiated:

Rule 5

```
achieve(p145,knowref(system,entity(1,Object)) ⇐
    mgoal(Hearer,Speaker,p145,knowref(system,entity(1,Object)) &
    evaluation(p145,success) &
    true(knowref(system,entity(1,Object)))
```

It is easy to see that the first two conditions hold, due to the beliefs that were just added. The third condition, however, is not as straightforward. In the plan derivation p145, Object is instantiated to antenna1. However, it is not instantiated in the mgoal proposition. Hence, our knowref predicate not only checks the contents of the discourse entity that it takes as a parameter, it also checks if there is a plan

p207: `accept-plan(p145)`
Constraint: `achieve(p145,knowref(system,entity(1,antenna1)))`
`plan(user,p145,knowref(system,entity(1,antenna1)))`
`s-accept(p145)`
Effect: `bel(user,goal(system,mb(system,user,achieve(p145,`
`knowref(system,entity(1,antenna1))))))`

Figure 6.8: Constructed Judgement Plan (p207)

derivation that contains this information. In this case there is, namely p145, and so knowref succeeds.[7] Thus the system adds the following belief:

```
achieve(p145,knowref(system,entity(1,antenna1)))
```

6.4 Constructing "Okay"

Since the system has inferred the referent of the refashioned referring expression, it can apply rule 9 and so adopts the goal of informing the user that it has identified the referent.

```
goal(system,mb(system,user,achieve(p145,
     knowref(system,entity(1,antenna1)))))
```

In figure 6.8, the plan derivation, p207, that was constructed to achieve this goal is shown. So, the system gives the generator the action `s-accept(p145)`, which would be realized by "Okay."

Lastly, the system adds the following beliefs about the constructed plan:

```
evaluation(p207,success)
plan(system,p207,mb(system,user,achieve(p145,knowref(system,
     entity(1,antenna1)))))
contributed(system,p207)
```

[7]This treatment is somewhat problematic, but it is due to our simplistic treatment of discourse entities.

Chapter 7

Implementation

In this chapter we outline our implementation, specifically, our plan construction algorithm, our plan inference algorithm, and our belief module. We also discuss the role of discourse entities in our implementation.

The system is implemented in Prolog, and we use Prolog variables to encode the variables that are in our plan schemas. By doing this, once a plan variable is instantiated, all co-referential variables are also instantiated. The use of Prolog variables has also influenced the manner in which we present our examples, for if a variable is instantiated, we show its value rather than its name.

7.1 Discourse Entities

We use discourse entities to represent the referring expression that is being collaborated upon; but more importantly, to indirectly represent the object that is being referred to. Discourse entities are represented by the predicate `entity(Var,Object)`, where `Object` represents the object being referred to and `Var` is a unique identifier.

Discourse entities are parameters of all surface speech actions that give rise to referring expressions, thus capturing the coherency among the actions. This becomes especially important when there is a referring expression embedded inside another referring expression. For instance, "the black bird in the cage" would correspond to the following surface speech actions:

```
s-refer(Entity1)
s-attrib(Entity1,λX·name(X,bird))
s-attrib(Entity1,λX·colour(X,brown))
s-attrib-rel(Entity1,Entity2,λX·λY·in(X,Y))
```

```
s-refer(Entity2)
s-attrib(Entity2,λX·name(X,cage))
```

As can be seen, the discourse entities, `Entity1` and `Entity2`, indicate which object that each surface speech action is about.

An advantage of using discourse entities, rather than a term representing the object itself, is that the relationship among the surface speech actions is captured without relying on using Prolog variables. Consider again the above referring expression. The input to our system would have `Entity1` instantiated to `entity(1,Obj1)` and `Entity2` instantiated to `entity(2,Obj2)`. Since these two discourse entities cannot be unified (1 and 2 cannot be unified), the plan recognition process can use standard Prolog unification when deriving a plan derivation.

7.2 Plan Construction

Before we discuss our plan construction algorithm, we need to give a more formal definition of a what a plan derivation is. A plan derivation can be defined recursively as follows. We start with a plan instance as the root of the derivation. For each plan instance in the derivation, each plan header in its decomposition is expanded into a plan instance, and the plan header is unified with the header of the plan instance. An instantiated derivation is a plan derivation in which all variables in the plan instances are instantiated, and all co-referential variables have the same value.

Motivated by Pollack (1986), we view an instantiated plan derivation not as a data structure, but as a mental object that an agent has beliefs about. The mental actions might not be executable and the constraints might not hold.[1] An agent will view an instantiated plan derivation as valid just in case it believes that all of its constraints hold and all of its mental actions are executable.

Now that this background has been given, we can state what our plan constructor does. Given an effect, the plan constructor finds an instantiated plan derivation that is valid (with respect to the planning agent's beliefs), and whose root plan achieves the effect. The yield of the plan derivation can then be given as input to a module that generates the surface form of the utterance.

[1] Since we assume that the plan library is shared and that agents can execute surface speech actions, we do not consider beliefs about generation or about the executability of primitive actions.

7.2.1 Plan Construction Heuristics

In constructing a plan derivation to accomplish an effect, there might be many possible plan derivations to choose from. This is particularly true when constructing a referring expression plan, since the plan schemas do not encode any preference for a description of minimal length. We have intentionally designed our plan schemas to exclude preference information. We feel that such information should be captured as general heuristics that the plan constructor uses in choosing one plan derivation over another. A benefit of this approach is that the heuristics can be applied to plan construction in general rather than having preference information for every plan schema. Also, it is not entirely clear how preference information should be used for plan inference.

In section 3.6, we proposed that one such planning heuristic would be to prefer plan derivations with the fewest number of primitive actions. This heuristic captures the preference of people to use descriptions of minimal length and thus avoid conversational implicature (Reiter, 1990).

There are other heuristics that could also be used. For instance, an agent not only has the goal that it is currently constructing a plan to achieve, it also has many other goals. Hence, plan derivations could be judged in terms of how well they contribute to satisfying these goals. Also, in constructing a plan for communicating with another agent, the speaker might have to make assumptions about the hearer's beliefs. A plan derivation that makes a minimal number of assumptions or that is based on the strongest beliefs would be preferable.

In implementing a planner that uses heuristics in choosing a plan, it is necessary to incorporate these heuristics as early in the construction process as possible. Otherwise there will be a combinatorial explosion of plan derivations to consider. In our implementation, the plan constructor orders the plan derivations by the number of primitive actions. At each step, it expands the plan derivation with the fewest number of primitive actions.

A side effect resulting from the way our plan schemas have been encoded is that plan derivations are not viewed as invalid on the basis of having duplicate primitive action instances. Since such derivations will not be chosen due to the heuristic, our plan constructor immediately removes them from consideration.

7.2.2 The Plan Construction Algorithm

The plan constructor is implemented using a best-first search strategy. It starts with the set of plan schemas that satisfy the effect that is to be achieved. It finds all

plan derivations whose constraints are satisfiable, and puts them into a list of partial plan derivations. Each partial plan derivation in the list has an associated list of unexpanded nodes and this list is initialized to its decomposition. The algorithm then repeatedly removes the partial plan derivation with the fewest number of primitive actions, finds all expansions of its leftmost unexpanded node, and adds these expansions (possibly none) to the list. This process is repeated until it finds a plan derivation that does not have any unexpanded nodes. Given below is the algorithm for expanding a node:

- **mental action**
  - Perform the action.[2]
- **primitive action**
  - Instantiate the where-predicates.
  - Ensure constraints are satisfiable (instantiate variables).
  - Ensure action is not already in the derivation.
- **plan header (that is not a primitive action)**
  - Instantiate the where-predicates.
  - Ensure constraints are satisfiable (instantiate variables).
  - Add decomposition to the front of the list of unexpanded nodes.

7.3 Plan Inference

In order for a hearer to understand an utterance, he must determine what the speaker intended by way of the utterance. In our model, this is accomplished by plan inference. By inferring the plan derivation that underlies the utterance, the hearer can determine the effect that the speaker intended the plan to achieve.

Since the speaker and hearer do not have the same beliefs, the hearer must be able to infer a plan even if he would view it as invalid. Following Pollack (1986), our plan inference process can infer invalid plans in which either a constraint does not hold or a mental action is not executable. Since we assume the plan library is shared and since the primitive actions have already been executed, we do not considered the other types of invalidities that Pollack considers.

[2]Mental actions are performed immediately. This allows the action to affect the rest of the construction process.

In inferring a plan derivation, we first try to find a plan derivation that accounts for the primitive actions that were observed. Second, we evaluate it by attempting to find an instantiation for the variables such that all of the constraints hold and the mental actions are satisfiable with respect to the hearer's beliefs about the speaker's beliefs. We refer to the first process as plan recognition, and the second process as plan evaluation.

In evaluating a plan derivation, we might not be able to find an instantiation for the variables.[3] In this case, the hearer will view the plan as invalid. However, as Pollack (1986, p. 5) points out, "simply knowing that a plan is invalid is not sufficient—agents must also be able to locate the source(s) of the invalidity." So, when a plan is found to be invalid, we determine the constraint or mental action that is the source of the invalidity.

7.3.1 Plan Recognition

The plan recognition process attempts to find a plan derivation whose yield is the set of primitive actions that was observed and whose root plan is a specially marked end plan.[4] It does not consider whether the constraints and mental actions are satisfiable, but it does unify the variables in the plan headers into the plan derivation. Hence plan derivations in which this unification cannot proceed are rejected.

The plan recognizer used in this system is based on chart parsing (see Allen 1987).[5] It has been modified to parse actions instead of words and we have removed the restriction that the input be ordered. So, the edges in our chart represent the set of primitive actions that is accounted for by the edge, rather than a start and end vertex of a list of words. A side effect of this is that primitive actions need to express any relationships with other actions that are implicit in the utterance. For referring expressions, this is captured by the use of discourse entities.

The parsing strategy is essentially bottom-up, breadth-first. There is an exception though. Some of the non-primitive plan schemas have a decomposition that consists of only mental actions (e.g., `relax-constraint`) or have a null decomposition (e.g., the `modifiers` schema for terminating the recursion). Since the parsing strategy is bottom-up starting from the primitive actions that were observed, plans of this type will never be added to the chart, and hence plan derivations that include them would not be properly recognized. Our solution to this problem is that whenever

[3] We will not consider the case where the plan recognition process is not able to find any plan derivation that accounts for the primitive actions.

[4] Our usage of end plans follows Kautz and Allen (1986).

[5] The actual code is based on the chart parser given by Ross (1989).

an active edge is added to the chart, a search is made to see if it has any leftmost descendents of this type. These descendents are then added as inactive entries, and normal bottom-up parsing can then connect them to the active edges that need them.

Applying parsing techniques to plan recognition is not a new idea. Sidner (1985) suggests the idea, and Vilain (1990) formalizes it in terms of chart parsing by recasting the work of Kautz and Allen (1986) on plan recognition. Unfortunately, Vilain shows that recognizing plans with abstraction and partial step order, which characterizes our work, is NP-complete.

7.3.2 Plan Evaluation

After a plan derivation has been recognized, the plan is evaluated. The plan evaluation algorithm tries to evaluate the constraints and mental actions. It prefers to evaluate them in the order that the plan constructor uses in constructing the plan derivation. However, the plan schemas have been formulated for plan construction, and there is a difference in the knowledge that the speaker has when constructing a plan to accomplish a certain goal and the knowledge that the hearer has when trying to infer the plan of another agent from the observed primitive actions. So, it might not be efficient or even possible to evaluate plan derivations in the order specified in the plan schemas. The plan evaluator takes this into account by using meta-level knowledge to choose the order in which to evaluate the constraints and mental actions in the plan derivation. This knowledge encodes which parameters of a predicate should be instantiated before it can be evaluated. Note that since predicates might include other predicates as parameters, the order of evaluation depends on all of the predicates in a constraint or mental action. For instance, for `bel(user,colour(Object,black))`, meta-knowledge about `bel` and `colour` is considered. Furthermore, the predicate

`subset(Set,`λ`X·Prop,Subset)`

requires special treatment. This predicate can be evaluated if `Set` is instantiated and `Prop`, with `X` instantiated to a dummy value, can be evaluated.

If the plan evaluator is not successful, one of the constraints or mental actions in the inferred plan derivation will be found to be unsatisfiable. Since the evaluator dynamically chooses the order in which to evaluate the constraints and mental actions, the one that it failed on might not be the cause of the error. Hence, when it finds an error, it tries to determine if the error is a result of a constraint or mental action whose evaluation was postponed. Our algorithm for determining the cause looks for syntactic similarities. All of the postponed constraints and mental actions, and the node that it failed on are stripped down to their base propositions. This involves stripping off

the operators `subset`, `mb`, `ub`, `bel`, and λ`X`. With these stripped propositions, the evaluator checks to see if any of the postponed constraints or mental actions unifies with the failed node. For instance, consider the situation where the evaluator finds that the following mental action fails:

```
subset([bird1,bird2],λX·ub(user,system,colour(X,black)),NewCand)
```

The evaluator would be able to determine that it was the following postponed constraint that was the cause of the error:

```
mb(user,system,colour(Object,black))
```

As mentioned in section 4.6 and section 5.1.3, the plan evaluator treats `construct` and `evaluate` specially. When evaluating `construct`, the evaluator knows the primitive actions that `construct` must account for and the goal it is trying to achieve. So, the evaluator invokes the plan recognizer rather than the plan constructor. For `evaluate`, the plan evaluator does not evaluate it.

The algorithm proceeds as follows: It first evaluates all of the where-predicates. Second, it gathers all of the constraints and mental actions into a list. Third, it iteratively chooses the first unevaluated constraint or mental action that has the required parameters instantiated as determined by the meta-knowledge, and tries to prove that it is satisfiable; this results in its variables being instantiated, as well as any co-referential variables in other constraints and mental actions being instantiated. If it fails in evaluating a constraint or mental action, then it determines which constraint or mental action is the cause of the failure.

7.4 Belief Module

We have incorporated a belief module into our system. This module provides the minimum functionality that is needed in order to support plan construction, plan inference, and the adoption of beliefs and goals. Two functions are provided, a function for adding a new belief, and a function for testing whether the system holds a belief. We do not provide a function for retracting a belief.

We represent our beliefs internally using the predicate `abel(N,Prop)` (Cohen and Levesque, 1990, p. 232), which is the `N`th alternating belief between the system and the user. The proposition `abel(1,Prop)` represents that the system believes `Prop`, `abel(2,Prop)` represents that the system believes the user believes `Prop`, etc. We do not allow the proposition inside `abel` to include any belief operators. Since `abel` is

the only predicate that we use for representing beliefs, we do not represent mutual belief, but as mentioned below, we have rules for inferring it.

When adding a new belief, the effect that this belief has on the other beliefs must be taken into consideration. This is especially true when adding the proposition `replace(Plan,NewPlan)` and `error(Plan,Node)`. For `replace`, if the system has a belief about an `mgoal` that has `Plan` as a parameter, `Plan` is replaced by `NewPlan`. For `error(Plan,Node)`, where `Node` is a constraint involving mutual belief, we could add evidence so that this mutual belief will not be inferred in the future. However, we do not address this issue.

When determining whether our system holds a belief, we use a number of inference rules. The first inference rule allows the system to assume that if the system believes a proposition then it is mutually believed. In particular, we can use this rule to derive that the system believes that the user believes `Prop` if the system believes it. This rule is intended to correspond with the co-presence heuristics of Clark and Marshall (1981) and it only applies to certain predicates, such as `mgoal`, `colour`, and `size`. This rule is obviously too strong, for it should ensure that there is no evidence to the contrary (see Nadathur and Joshi 1983).

The second inference rule is used to ascribe beliefs based on other beliefs that the system has. This includes inferring that the system and the user have a `plan` if they have an `mgoal`.

A third inference rule is used to allow the system to assume a belief based on no evidence. As with the first rule, this rule applies only to certain predicates. In this case, it allows the system to believe that the user believes that there is an error in a plan derivation. Note that without this inference rule, the system would find an inferred judgement plan to be invalid.

A fourth inference rule sanctions the belief that `Prop` holds if the proposition `relax(Prop1,Prop)` is believed, and `Prop1` is also believed.

Chapter 8

Conclusion

We have presented a computational model of how a conversational participant collaborates in making a referring expression. Our work is based on the model that Clark and Wilkes-Gibbs (1986) propose, augmented with the work done by Clark and Schaefer (1989) on contributing to discourse. In recasting their work into a computational model, we have adopted the view that language is goal-oriented behaviour. This has allowed us to do the following. First, account for the tasks of building a referring expression and identifying its referent by using plan construction and plan recognition. Second, account for the conversational moves that participants make during the acceptance process by using meta-plans. Third, show how beliefs are adopted from an inferred plan, which in turn gives rise to the adoption of discourse goals to further the collaborative activity.

Although our work has focused on referring expressions, we feel that it is relevant to collaboration in general and to how agents contribute to discourse. So, in the next section, we comment on what it means for two agents to be collaborating in building a plan, and how this mode of interaction differs from viewing one of the agents as an information provider. Then in the following section, we return to the discussion in section 5.1.2 on judging contributions. In the renewed discussion, we propose a formalization of the criteria for accepting contributions in terms of discourse intentions.

Then in the remainder of the chapter, we discuss a few assumptions that we have made in our work that are worth noting; we compare our work to Litman and Allen (1987) and to Grosz and Sidner (1990); and we conclude by pointing our some directions for future research.

8.1 Collaboration

There are many ways that agents interact in conversation. In this thesis, we have looked at how agents collaborate in making a referring expression, which we have equated with collaborating in building a plan. But what does it mean for two agents to collaborate in building a plan? From the work of Clark and Wilkes-Gibbs, we have identified this with having mutual responsibility in building a plan to achieve a goal, and where the agents interact by judging and refashioning the current plan.

Before exploring this further, let's re-examine dialogue (5.1), reprinted here as dialogue (8.1).

(8.1) **P:** [1] The 8:50 to Montreal?
C: [2] 8:50 to Montreal. Gate 7.
P: [3] Where is it?
C: [4] Down this way to your left. Second one on the left.
P: [5] OK. Thank you.

In this dialogue, the passenger has a plan to board a train. However, her plan is underspecified, and hence she requests information from the clerk and has an expectation that he will answer her request. However, the passenger and the clerk are not collaborating in building this plan. The passenger is simply using the clerk as a resource for obtaining information, and so she is controlling the conversation (see Walker and Whittaker, 1990). Note that the clerk might even infer the passenger's plan, but he does this in order to provide additional context for her question, or to be able to anticipate any obstacles in her plan (Allen, 1979).[1]

So, there are different ways that agents can interact with respect to building a plan. In the case of referring expressions, the agents are mutually responsible for achieving the goal. We feel that this allows the agents to interact such that neither agent assumes control of the dialogue, thus allowing both agents to contribute to the best of their ability without being guided or impeded by the other agent. This results in the agents minimizing their combined effort in achieving the goal. This type of interaction is not possible in dialogue (8.1), since the agents do not have the plan in their common ground, nor are they necessarily mutually responsible. Additionally,

[1] This of course means that the passenger must be able to understand any helpful comments from the clerk. Hence the passenger should be able to understand the clerk's response with respect to her underlying plan, rather than just with respect to the request. For instance, the passenger, after uttering "Where is it?", should be able to understand a response such as "It is the only gate" or "Follow that person."

it would not be expedient for the agents to collaborate in the building of the plan, since the passenger is perfectly able to build the plan herself—she simply needs some information.

8.2 Accepting Contributions

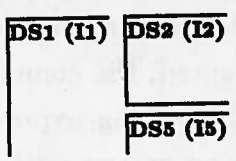
In providing a computational model of how agents collaborate in making a referring expression, we have adopted the view of Clark and Schaefer (1989) that the judgement and refashioning moves are also contributions and hence are judged in accordance with whether the hearer "believes he is understanding well enough for current purposes" (p. 267). In section section 5.1.2, we gave criteria for determining whether judgements and refashionings are understood that do not depend on whether the refashioned referring expression is acceptable. Hence we distinguish between the acceptance of a refashioning move and the acceptance of the refashioned referring expression. The outcome of this is that when a refashioning move is accepted, the common ground of the participants is updated to reflect the refashioned referring expression, thus allowing it to be judged and refashioned in the same way as the initial referring expression, as Clark and Wilkes-Gibbs (1986) claim.

We have relied on Clark and Schaefer's proposal that a hearer accepts a contribution depending on whether he has understood it well enough for current purposes. But, in general, what is the current purpose that they allude to? We feel that a key to understanding this is to relate their work to that done by Grosz and Sidner (1986) on intentions and discourse structure. Utterances have purposes associated with them, and these utterances are embedded in discourse segments that also have a purpose. So, in determining whether to accept a contribution, it is only the purpose that is associated with the contribution that comes into play in determining whether to accept it. Note that an utterance might be intended to fulfill the purpose of the segment that it is embedded in. But if it doesn't, then this is not grounds for rejecting the contribution, but only grounds for not ending that discourse segment.

With this interpretation of Clark and Schaefer's criteria, let's re-examine the acceptance process for a referring expression. First, since there is an intention underlying a referring expression, namely for the responder to identify the referent, we will assume that the utterance of a referring expression initiates a new discourse segment, say `DS1`. After a referring expression has been uttered, if the responder cannot identify the referent then he will not accept it, since his current purpose is to understand the referent. The two participants will hence make judgements and refashionings, and these are discourse segments embedded inside segment `DS1`. The purpose of these seg-

ments is not the same as the purpose of DS1. For a judgement move, it is to inform the other participant of a judgement, and for a refashioning move, it is to inform the other of a replacement referring expression. Hence, they are accepted and so their discourse segments are ended independently of the acceptance of the current referring expression. So, subsequent judgements and refashionings are embedded directly in DS1. Once the conversational participants accept the referring expression, not only is the acceptance accepted, but it also causes the current discourse segment, DS1, to end.

In section 5.1.1, we examined subdialogue (5.1), repeated as (8.1), to illustrate the relevance of Clark and Schaefer's work. In particular, we claimed that the third utterance implicitly accepts the second utterance and so is not a contribution to accept it. Let's now re-examine it in the light of discourse segments and intentions.



P: 1 The 8:50 to Montreal?

C: 2 8:50 to Montreal. Gate 7.

P: 3 Where is it?

Figure 8.1: Discourse Segments for Dialogue (5.1)

In figure 8.1, we have reprinted the first three utterances, and indicated what we think is the discourse structure of the dialogue. For the first utterance, we claim that there are three distinct intentions—I1, I2, and I3—underlying it.

I1: P intends to know the location of the train so that she can board it.

I2: P intends C to inform her of the location of the train.

I3: P intends C to know that she wants him to inform her of the location of the train.

Note that I1 dominates I2, and I2 dominates I3. As shown in figure 8.1, this gives rise to three discourse segments, DS1, DS2, and DS3, corresponding to I1, I2, and I3, respectively.

Now the second utterance, which is itself a discourse segment, DS4, is being used to achieve C's intention that P know that the train is at gate 7. So, this utterance implicitly accepts the first utterance, and so signals the end of DS3. Note that DS4 is embedded inside DS2, since I2 dominates I4. The third utterance is the controversial one with regard to whether or not it signals the end of DS2 and DS4. We claim that it does, since the third utterance shows that P has understood the second utterance

well enough for the purposes I2 and I4. Hence, her utterance signals the acceptance of the second utterance. However, since she doesn't know the location well enough to board the train, her utterance is embedded inside of DS1.

8.3 A Few Assumptions

In this section, we discuss a few assumptions that we have made in carrying out our research that we feel are worth noting.

8.3.1 Initiator and Responder

In collaborating to make a referring expression, the initiator and the responder have different roles. The initiator knows the identify of the referent and the responder is trying to identify it. So, if the initiator refashions a referring expression, the utterance can be interpreted as "The one that ...", but if the responder refashions, the utterance can be interpreted as "Do you mean the one that ... ?"

Clark and Wilkes-Gibbs (1986), in their model of the acceptance process, do not distinguish between the initiator and the responder. Hence, it is unclear how their model accounts for the difference in the surface forms between the initiator and the responder's utterances. Our model also doesn't account for the difference in the forms of the utterances. In fact, the same meta-plans and surface speech acts are used by both agents in judging and refashioning referring expressions.

One avenue that is worth pursuing is the use of *try markers*. Clark and Wilkes-Gibbs propose that *try markers* (rising intonation) are used by a speaker to express her confidence in an utterance. Perhaps this can account for the difference between the initiator's utterances and the responder's, since the responder would not be as confident as the initiator would be that his refashioned referring expression identifies the referent. In our model, we have not addressed speaker confidence, and so it is difficult to say whether it can adequately account for the above phenomena.

8.3.2 Surface Speech Acts

We have postulated the following surface speech actions:

Referring:

`s-refer`

Attributing:

`s-attrib`, `s-attrib-rel`, and `s-attrib-rel-group`

Expressing judgement:

`s-accept`, `s-reject`, and `s-postpone`

Refashioning:

`s-expand`

Although we could have viewed these as acts of informing, this would have shifted the complexity into the parameter of the `inform` and it is unclear whether anything would have been gained. Instead, we feel that a parser can determine the surface speech actions, especially if it has access to a model of the discourse to help it disambiguate the utterance. Additionally, it should be easier for the generator to determine an appropriate surface form.

8.3.3 Presentations

Clark and Wilkes-Gibbs (1986) and Clark and Schaefer (1989) talk about presentations of a contribution, as well as judgements and refashionings. We have viewed the presentation of a contribution as being implicit. Hence, the effect of our refer plan is not to inform the hearer of a referring expression plan, but to have the hearer believe that the speaker has the goal for the hearer of identifying the referent. We assume that the hearer doesn't need to know that the speaker has presented a plan, because this will be obvious if he has understood the utterance.

8.4 Comparisons to Other Work

In this section, we compare our work to Litman and Allen (1987), and Grosz and Sidner (1990). We have limited ourselves to these works because we feel that they come closest to offering the framework for collaborative behaviour that we propose.

8.4.1 Litman and Allen

Litman and Allen's work on understanding clarification subdialogues has influenced our work greatly. It has provided us with the idea of using meta-plans to account for referring expressions that can be reasoned about and manipulated in the planning paradigm.

There are several major differences between our work and theirs. First, our work extends theirs, since it also accounts for the construction of an utterance and how discourse goals arise from inferring a plan. Second, we account for collaboration. So, our discourse plans are different from theirs, and we model an agent's discourse

expectations by way of intentions rather than a more rigid plan stack. Third, we take into account how plans are updated to reflect the contributions of the participants, a problem that we feel causes Litman and Allen to improperly recognize when a contribution is accepted.

8.4.2 Grosz and Sidner

Grosz and Sidner (1990)[2] are interested in the type of plans that underlie discourse where the agents are collaborating in order to achieve some goal. They propose that agents are building a shared plan. Their definition of a shared plan is in terms of the mutual beliefs and intentions of the participants regarding the actions that comprise the plan. They also discuss partial shared plans, which they define as follows: "First, [a partial shared plan] may contain only some of the full collection of beliefs and intentions of its associated full shared plan. Second, some of the beliefs included in it may be only partially specified ..." (Grosz and Sidner, 1990, p. 431). They then use this to show how their partial shared plans are updated by recognizing the beliefs and intentions underlying an utterance.

Our model differs from theirs in two important aspects.[3] First, not only do agents have a collection of beliefs and intentions regarding the actions of a (partial) shared plan, we feel that they also have an intention about the goal. It is this intention in conjunction with the partial shared plan that sanctions the adoption of beliefs and intentions about potential actions that will contribute to the goal, rather than merely the partial shared plan as indicated by their second conversational default rule (CDR2).

Second, we feel that their definition of a partial shared plan is too restrictive. They require, in order for an action to be part of a partial shared plan, that both agents believe (in fact mutually believe) that an action *contributes* to the goal, where the action contributes by either *enabling* or *generating*[4] the goal. However, this is too strong. In collaborating to achieve a mutual goal, participants sometimes propose an action that is not believed by the other participant or even the participant that is proposing it. By failing to represent such states, their model will be unable to represent the intermediate states in which a hearer might have understood how the

[2]See also the subsequent work by Lochbaum, Grosz, and Sidner (1990) and Lochbaum (1991).

[3]Their work also addresses a number of issues that we have not concerned ourselves with. First, we do not need to represent that an agent intends to perform an action or that it is able to execute the action (because the actions have already been performed). Second, we assume that agents have complete knowledge of the plan schemas and that this knowledge is shared.

[4]They consider several types of generation to account for simultaneous actions, consecutive actions, etc.

speaker's utterance contributes to a plan, but doesn't agree with it. In section 5.1.1 we showed that this is important, since if a refashioning is understood, the common ground is updated, and we feel that this should correspond to updating the shared plan. Furthermore, even the final plan might contain an action that is not mutually believed to be executable, as evidenced by an utterance such as "I don't think it will work, but let's try it anyways." Perhaps the only requirement should be that the plan is not incoherent.

8.5 Future Direction

There are many ways that this research can be extended. Perhaps the most obvious would be to extend the coverage of the model. We have examined one type of referring expression. We feel that our approach can be extended to handle references to objects in focus, references to objects outside of Appelt's *shared concept activation with identification intention* (1985c), and references in which the speaker is uncertain about the adequacy of the constructed referring expression. Also, the meta-plans that we have given do not handle all types of judgement and refashionings of referring expressions. Hence, our coverage of them could be expanded and even generalized so as to handle plans other than just referring expressions. By extending our model in this way, we should be able to account for our benchmark dialogue, dialogue (6.2) given in chapter 6, without simplifying it.

A second area for future research is to incorporate a better model of an agent's beliefs. When agents collaborate, their beliefs of the other agent as well as their own beliefs are being revised. This belief revision not only facilitates subsequent references, but it is also an integral part of the acceptance process, for the agents are acquiring beliefs that facilitate them in making a referring expression acceptable to both participants. However, in this thesis we have left this aspect practically unexplored.

A third area would be to further investigate collaborative behaviour. In this thesis, we have touched upon a number of questions that need to be answered more fully. First, what does it mean to say that two agents are collaborating, and what behavioral effects will be manifested? Second, how does this mode of interaction differ from other modes, such as how a master and an apprentice interact? Third, how do agents establish a mode of interaction based on collaboration? Fourth, what reasons do agents consider in deciding whether to establish such a mode? By answering these questions, we will not only have a better model to base natural language interfaces on, but we will also have a better understanding of how people interact.

Bibliography

Allen, J. (1987). *Natural Language Understanding.* Benjamin/Cummings, Menlo Park.

Allen, J. F. (1979). A plan-based approach to speech act recognition. Doctoral dissertation, Technical Report 131, Department of Computer Science, University of Toronto.

Allen, J. F. and Perrault, C. R. (1980). Analyzing intention in utterances. *Artificial Intelligence*, 15:143–178. Reprinted in (Grosz, Sparck-Jones and Webber, 1986).

Appelt, D. and Kronfeld, A. (1987). A computational model of referring. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '87)*, pages 640–647.

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

Appelt, D. E. (1985b). *Planning English Sentences.* Cambridge University Press.

Appelt, D. E. (1985c). Some pragmatic issues in the planning of definite and indefinite noun phrases. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 198–203.

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

Clark, H. H. and Brennan, S. E. (1990). Grounding in communication. In Resnick, L., Levine, J., and Behreno, S., editors, *Perspectives on Socially Shared Cognition.* APA.

Clark, H. H. and Marshall, C. R. (1981). Definite reference and mutual knowledge. In Joshi, A. K., Webber, B. L., and Sag, I., editors, *Elements of Discourse Understanding*, pages 10–62. Cambridge University Press, Cambridge.

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

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

Cohen, P. R. (1981). The need for referent identification as a planned action. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 31–36.

Cohen, P. R. (1984). Referring as requesting. In *Proceedings of the 10th International Conference on Computational Linguistics (COLING '84)*, pages 207–211.

Cohen, P. R. and Levesque, H. J. (1990). Rational interaction as the basis for communication. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, SDF Benchmark Series, pages 221–255. MIT Press.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212. Reprinted in (Grosz, Sparck-Jones and Webber, 1986).

Dale, R. (1989). Cooking up referring expressions. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 68–75.

Garrod, S. and Anderson, A. (1987). Saying what you mean in dialogue: A study in conceptual and semantic co-ordination. *Cognition*, 27:181–218.

Goodman, B. A. (1985). Repairing reference identification failures by relaxation. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 204–217.

Grice, H. P. (1957). Meaning. *Philosophical Review*, 66:377–388.

Grosz, B. J. (1981). Focusing and description in natural language dialogues. In Joshi, A. K., Webber, B. L., and Sag, I., editors, *Elements of Discourse Understanding*, pages 84–105. Cambridge University Press, Cambridge.

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

Grosz, B. J. and Sidner, C. L. (1990). Plans for discourse. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, SDF Benchmark Series, pages 417–444. MIT Press.

Grosz, B. J., Sparck-Jones, K., and Webber, B. L., editors (1986). *Readings in Natural Language Processing.* Morgan Kaufmann Publishers.

Horrigan, M. K. (1977). Modelling simple dialogs. Master's Thesis, Technical Report 108, Department of Computer Science, University of Toronto.

Kautz, H. A. and Allen, J. F. (1986). Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI '86)*, pages 32–37.

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

Litman, D. J. (1985). Plan recognition and discourse analysis: an integrated approach for understanding dialogues. Doctoral dissertation, Technical Report 170, Department of Computer Science, University of Rochester.

Litman, D. J. and Allen, J. F. (1987). A plan recognition model for subdialogues in conversations. *Cognitive Science*, 11(2):163–200.

Lochbaum, K. E. (1991). An algorithm for plan recognition in collaborative discourse. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 33–38.

Lochbaum, K. E., Grosz, B. J., and Sidner, C. L. (1990). Models of plans to support communication: An initial report. In *Proceedings of the National Conference on Artificial Intelligence (AAAI '90)*, pages 485–490.

Mellish, C. S. (1985). *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood Series in Artificial Intelligence. Ellis Horwood, Chichester, West Sussex, England.

Nadathur, G. and Joshi, A. K. (1983). Mutual beliefs in conversational systems: Their role in referring expressions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '83)*, pages 603–605.

Perrault, C. R. (1990). An application of default logic to speech act theory. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, SDF Benchmark Series, pages 161–185. MIT Press.

Perrault, R. and Cohen, P. R. (1981). It's for your own good: a note on inaccurate reference. In Joshi, A. K., Webber, B. L., and Sag, I., editors, *Elements of Discourse Understanding*, pages 217–230. Cambridge University Press, Cambridge.

Pollack, M. E. (1986). Inferring domain plans in question-answering. Technical Note 403, SRI International. Also Doctoral dissertation, University of Pennsylvania, Philadelphia, PA.

Pollack, M. E. (1990). Plans as complex mental attitudes. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, SDF Benchmark Series, pages 77–103. MIT Press.

Ramshaw, L. A. (1991). A three-level model for plan exploration. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 39–46.

Reiter, E. (1990). The computational complexity of avoiding conversational implicature. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 97–104.

Ross, P. (1989). *Advanced Prolog: Techniques and Examples.* International series in logic programming. Addison Wesley, Wokingham, England.

Searle, J. R. (1969). *Speech acts: An essay in the philosophy of language.* Cambridge University Press, Cambridge.

Searle, J. R. (1990). Collective intentions and actions. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, SDF Benchmark Series, pages 401–415. MIT Press.

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

Svartvik, J. and Quirk, R. (1980). *A Corpus of English Conversation.* Lund Studies in English. 56. C.W.K. Gleerup, Lund.

Vilain, M. (1990). Getting serious about parsing plans: a grammatical analysis of plan recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI '90)*, pages 190–197.

Walker, M. and Whittaker, S. (1990). Mixed initiative in dialogue: An investigation into discourse segmentation. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 70–78.

Webber, B. L. (1983). So what can we talk about now? In Brady, M. and Berwick, R. C., editors, *Computational Models of Discourse*, pages 331–371. MIT Press, Cambridge.

Appendix A

Trace of the System

In this appendix, we give the full trace of the example that was presented in chapter 6. In this example, the system takes the role of the responder, person B, in dialogue (6.1), repeated below as dialogue (A.1).

(A.1) **A:** [1] See the weird creature.
B: [2] In the corner?
A: [3] No, on the television.
B: [4] Okay.

In the trace, lambda expressions, $\lambda X \cdot Prop$, are represented as:

```
X^Prop
```

Propositions of the form $(\lambda X \cdot Prop)(Y)$ are represented as:

```
apply(X^Prop,Y)
```

Prolog variables are represented as _n, where n is an integer. Lines in the trace have been formatted so that they will fit on the page. Where a line has been broken, a dash followed by a space is used to start the continuation. Plan derivations and lists of actions have been formatted for readability. When they are a parameter in a proposition, they are represented as a variable of the form Vn, where n is an integer. On a line following the proposition, the value of the variable is given. Section titles, which correspond to the section titles of chapter 6, have been inserted into the trace.

A.1 Initialization

```
User Model: Adding Proposition
category(fern1,creature)

User Model: Adding Proposition
assessment(fern1,weird)

User Model: Adding Proposition
in(fern1,corner1)

User Model: Adding Proposition
category(antenna1,creature)

User Model: Adding Proposition
assessment(antenna1,weird)

User Model: Adding Proposition
on(antenna1,television1)

User Model: Adding Proposition
category(corner1,corner)

User Model: Adding Proposition
category(television1,television)

User Model: Adding Proposition
world([fern1,antenna1,corner1,television1])
```

A.2 Understanding "The weird creature"

```
==============
Understanding
==============

---------------
Plan Recognizer
---------------

Actions = s-refer(entity(1,_95))
          s-attrib(entity(1,_95),_115^assessment(_115,weird))
          s-attrib(entity(1,_95),_126^category(_126,creature))

Initial Setup
-------------
Adding inactive edge:
Head: s-refer(entity(1,_28515))
Acts: s-refer(entity(1,_28515))

Adding active edge:
Head: refer(_33753)
Need: s-refer(_33753)

Adding config (from active edge):
Active:   refer(_33753)
Inactive: s-refer(entity(1,_28515))

Adding inactive edge:
Head: s-attrib(entity(1,_28515),_28525^assessment(_28525,weird))
Acts: s-attrib(entity(1,_28515),_28525^assessment(_28525,weird))

Adding active edge:
Head: modifier(_46979,_46980,_46981)
Need: s-attrib(_46979,_47019)

Adding config (from active edge):
```

```
Active:   modifier(_46979,_46980,_46981)
Inactive: s-attrib(entity(1,_28515),_28525^assessment(_28525,weird))

Adding inactive edge:
Head: s-attrib(entity(1,_28515),_28539^category(_28539,creature))
Acts: s-attrib(entity(1,_28515),_28539^category(_28539,creature))

Adding config (from inactive edge):
Active:   modifier(_46979,_46980,_46981)
Inactive: s-attrib(entity(1,_28515),_28539^category(_28539,creature))

Adding active edge:
Head: headnoun(_63427,_63428)
Need: s-attrib(_63427,_63462^category(_63462,_63470))

Adding config (from active edge):
Active:   headnoun(_63427,_63428)
Inactive: s-attrib(entity(1,_28515),_28539^category(_28539,creature))

Adding active edge:
Head: modifier(_63357,_63358,_63359)
Need: s-attrib(_63357,_63397)

***duplicate active edge (removed)***

End of Initial Setup
--------------------

Applying config (3 left, step 1)
--------------------------------
Active:   refer(_33753)
Inactive: s-refer(entity(1,_28515))

Adding active edge:
Head: refer(entity(1,_93284))
Need: describe(entity(1,_93284))
Acts: s-refer(entity(1,_93284))

Applying config (2 left, step 2)
--------------------------------
Active:   modifier(_46979,_46980,_46981)
Inactive: s-attrib(entity(1,_28515),_28525^assessment(_28525,weird))

Adding inactive edge:
Head: modifier(entity(1,_105818),_105652,_105653)
Acts: s-attrib(entity(1,_105818),_105820^assessment(_105820,weird))

Adding active edge:
Head: modifiers(_112377,_112378)
Need: modifier(_112377,_112378,_112403)

Adding config (from active edge):
Active:   modifiers(_112377,_112378)
Inactive: modifier(entity(1,_105818),_105652,_105653)

Applying config (2 left, step 3)
--------------------------------
Active:   modifier(_46979,_46980,_46981)
Inactive: s-attrib(entity(1,_28515),_28539^category(_28539,creature))

Adding inactive edge:
Head: modifier(entity(1,_126854),_126688,_126689)
Acts: s-attrib(entity(1,_126854),_126856^category(_126856,creature))

Adding config (from inactive edge):
Active:   modifiers(_112377,_112378)
Inactive: modifier(entity(1,_126854),_126688,_126689)
```

```
Adding active edge:
Head: modifiers(_133413,_133414)
Need: modifier(_133413,_133414,_133439)

***duplicate active edge (removed)***

Applying config (2 left, step 4)
--------------------------------
Active:   headnoun(_63427,_63428)
Inactive: s-attrib(entity(1,_28515),_28539^category(_28539,creature))

Adding inactive edge:
Head: headnoun(entity(1,_150474),_150300)
Acts: s-attrib(entity(1,_150474),_150334^category(_150334,creature))

Adding active edge:
Head: describe(_156710)
Need: headnoun(_156710,_156733)

Adding config (from active edge):
Active:   describe(_156710)
Inactive: headnoun(entity(1,_150474),_150300)

Applying config (2 left, step 5)
--------------------------------
Active:   modifiers(_112377,_112378)
Inactive: modifier(entity(1,_105818),_105652,_105653)

Adding active edge:
Head: modifiers(entity(1,_168821),_168669)
Need: modifiers(entity(1,_168821),_168694)
Acts: s-attrib(entity(1,_168821),_168864^assessment(_168864,weird))

***Topdown addition of inactive entries (1)***

Adding inactive edge:
Head: modifiers(entity(1,_177235),_177232)

Adding config (from inactive edge):
Active:   modifiers(entity(1,_168821),_168669)
Inactive: modifiers(entity(1,_177235),_177232)

Applying config (2 left, step 6)
--------------------------------
Active:   modifiers(_112377,_112378)
Inactive: modifier(entity(1,_126854),_126688,_126689)

Adding active edge:
Head: modifiers(entity(1,_190889),_190737)
Need: modifiers(entity(1,_190889),_190762)
Acts: s-attrib(entity(1,_190889),_190932^category(_190932,creature))

Adding config (from active edge):
Active:   modifiers(entity(1,_190889),_190737)
Inactive: modifiers(entity(1,_177235),_177232)

***Topdown addition of inactive entries (1)***

Adding inactive edge:
Head: modifiers(entity(1,_199303),_199300)

***duplicate inactive edge***

Applying config (2 left, step 7)
--------------------------------
Active:   describe(_156710)
Inactive: headnoun(entity(1,_150474),_150300)
```

```
Adding active edge:
Head: describe(entity(1,_217432))
Need: modifiers(entity(1,_217432),_217306)
Acts: s-attrib(entity(1,_217432),_217478^category(_217478,creature))

Adding config (from active edge):
Active:   describe(entity(1,_217432))
Inactive: modifiers(entity(1,_177235),_177232)

***Topdown addition of inactive entries (1)***

Adding inactive edge:
Head: modifiers(entity(1,_225522),_225519)

***duplicate inactive edge***

Applying config (2 left, step 8)
--------------------------------
Active:   modifiers(entity(1,_168821),_168669)
Inactive: modifiers(entity(1,_177235),_177232)

Adding inactive edge:
Head: modifiers(entity(1,_244291),_244288)
Acts: s-attrib(entity(1,_244291),_244351^assessment(_244351,weird))

Adding config (from inactive edge):
Active:   describe(entity(1,_217432))
Inactive: modifiers(entity(1,_244291),_244288)

Adding config (from inactive edge):
Active:   modifiers(entity(1,_190889),_190737)
Inactive: modifiers(entity(1,_244291),_244288)

Applying config (3 left, step 9)
--------------------------------
Active:   modifiers(entity(1,_190889),_190737)
Inactive: modifiers(entity(1,_177235),_177232)

Adding inactive edge:
Head: modifiers(entity(1,_265635),_265632)
Acts: s-attrib(entity(1,_265635),_265695^category(_265695,creature))

Adding config (from inactive edge):
Active:   modifiers(entity(1,_168821),_168669)
Inactive: modifiers(entity(1,_265635),_265632)

Applying config (3 left, step 10)
---------------------------------
Active:   describe(entity(1,_217432))
Inactive: modifiers(entity(1,_177235),_177232)

Adding inactive edge:
Head: describe(entity(1,_282269))
Acts: s-attrib(entity(1,_282269),_282331^category(_282331,creature))

Adding config (from inactive edge):
Active:   refer(entity(1,_93284))
Inactive: describe(entity(1,_282269))

Applying config (3 left, step 11)
---------------------------------
Active:   describe(entity(1,_217432))
Inactive: modifiers(entity(1,_244291),_244288)

Adding inactive edge:
Head: describe(entity(1,_297936))
Acts: s-attrib(entity(1,_297936),_297998^category(_297998,creature))
      s-attrib(entity(1,_297936),_298248^assessment(_298248,weird))
```

```
Adding config (from inactive edge):
Active:   refer(entity(1,_93284))
Inactive: describe(entity(1,_297936))

Applying config (3 left, step 12)
---------------------------------
Active:   modifiers(entity(1,_190889),_190737)
Inactive: modifiers(entity(1,_244291),_244288)

Adding inactive edge:
Head: modifiers(entity(1,_317235),_317232)
Acts: s-attrib(entity(1,_317235),_317295^category(_317295,creature))
      s-attrib(entity(1,_317235),_317551^assessment(_317551,weird))

Applying config (2 left, step 13)
---------------------------------
Active:   modifiers(entity(1,_168821),_168669)
Inactive: modifiers(entity(1,_265635),_265632)

Adding inactive edge:
Head: modifiers(entity(1,_332713),_332710)
Acts: s-attrib(entity(1,_332713),_332773^assessment(_332773,weird))
      s-attrib(entity(1,_332713),_333029^category(_333029,creature))

Applying config (1 left, step 14)
---------------------------------
Active:   refer(entity(1,_93284))
Inactive: describe(entity(1,_282269))

Adding inactive edge:
Head: refer(entity(1,_347469))
Acts: s-refer(entity(1,_347469))
      s-attrib(entity(1,_347469),_347719^category(_347719,creature))

Applying config (0 left, step 15)
---------------------------------
Active:   refer(entity(1,_93284))
Inactive: describe(entity(1,_297936))

Adding inactive edge:
Head: refer(entity(1,_360271))
Acts: s-refer(entity(1,_360271))
      s-attrib(entity(1,_360271),_360521^category(_360521,creature))
      s-attrib(entity(1,_360271),_360628^assessment(_360628,weird))

*** Agenda Empty ***

Number of complete plans found: 1

------------------------
Consider Plan Derivation
------------------------

refer(entity(1,_377109))
+-s-refer(entity(1,_377109))
+-describe(entity(1,_377109))
  +-headnoun(entity(1,_377109),_377163)
  | +-s-attrib(entity(1,_377109),_377197^category(_377197,creature))
  | +-subset(_377178,_377197^ub(_377170,_377174,category(_377197,creature)),_377163)
  +-modifiers(entity(1,_377109),_377163)
    +-modifier(entity(1,_377109),_377163,_377274)
    | +-s-attrib(entity(1,_377109),_377304^assessment(_377304,weird))
    | +-subset(_377163,_377341^ub(_377281,_377285,apply(_377304^assessment(_377304,weird),_377341)),
    |    - _377274)
    +-modifiers(entity(1,_377109),_377274)
      +-null
```

```
--------
Evaluate
--------

postponing evaluation of mb(system,user,category(_377109,creature))
<ma>  subset([fern1,antenna1,corner1,television1],_377197^ub(system,user,category(_377197,creature)),
       - _377163)
postponing evaluation of mb(system,user,category(_377109,creature))
postponing evaluation of mb(system,user,assessment(_377109,weird))
<ma>  subset([antenna1,fern1],_377341^ub(system,user,assessment(_377341,weird)),_377274)
postponing evaluation of mb(system,user,category(_377109,creature))
postponing evaluation of mb(system,user,assessment(_377109,weird))
<con> [antenna1,fern1]=[_377109]
 *** failed ***

Failure Evaluation
------------------
Failure Mode: [antenna1,fern1]=[_377109]


Inferred Plan
-------------

_414955:refer(entity(1,_377109))
+-_415331:s-refer(entity(1,_377109))
+-_415567:describe(entity(1,_377109))
  +-_415761:headnoun(entity(1,_377109),[antenna1,fern1])
  | +-_416228:s-attrib(entity(1,_377109),_377197^category(_377197,creature))
  | +-_416828:subset([fern1,antenna1,corner1,television1],_377197^ub(user,system,category(_377197,
  |    - creature)),[antenna1,fern1])
  +-_416961:modifiers(entity(1,_377109),[antenna1,fern1])
    +-_417155:modifier(entity(1,_13141),[antenna1,fern1],[antenna1,fern1])
    | +-_13507:s-attrib(entity(1,_13141),_13235^assessment(_13235,weird))
    | +-_13547:subset([antenna1,fern1],_13262^ub(user,system,apply(_13235^assessment(_13235,weird),
    |    - _13262)),[antenna1,fern1])
    +-_13552:modifiers(entity(1,_13141),[antenna1,fern1])
      +-_13582:null

User Model: Adding Proposition
plan(user,p1,knowref(system,entity(1,_13141)))

User Model: Adding Proposition
contributed(user,p1)

User Model: Adding Proposition
evaluation(p1,failure(p28))

===============
Checking Rules
===============
Adding beliefs about the inferred plan

Applying rule1
User Model: Adding Proposition
mgoal(system,user,p1,knowref(system,entity(1,_61241)))

Applying rule6
User Model: Adding Proposition
error(p1,p28)
```

A.3 Constructing "In the corner?"

```
==============
Checking Rules
==============
Adding goals
```

```
Applying rule7
User Model: Adding Proposition
goal(system,mb(system,user,error(p1,p28)))

==========
Responding
==========

Goal   : mb(system,user,error(p1,p28))
Effect: bel(user,goal(system,mb(system,user,error(p1,p28))))

---------
Construct
---------

Initial Setup
-------------
Trying: reject-plan(p1)
<con> error(p1,p28)
<con> plan(user,p1,_86268)
<con> constraint(p1,_86273,p28)
<con> yield(p1,p26,_86280)
<con> length(□,1)
*** failed ***
Trying: postpone-plan(p1)
<con> error(p1,p28)
<con> plan(user,p1,_86268)
<con> constraint(p1,_86273,p28)
<con> content(p1,p26,_86280)
<con> yield(p1,p26,□)
<con> modifiers(entity(1,_99601),[antenna1,fern1])=modifiers(_86293,_86294)
New edge added

End of Initialization
---------------------

Expanding (step 0  active 1)
----------------------------
Head: postpone-plan(p1)
Need: s-postpone(p1)
Trying: s-postpone(p1)
New edge added


Constructed Plan
----------------
postpone-plan(p1)
+-s-postpone(p1)

Actions to be fed to generator
------------------------------
s-postpone(p1)

User Model: Adding Proposition
plan(system,p31,mb(system,user,error(p1,p28)))

User Model: Adding Proposition
contributed(system,p31)

User Model: Adding Proposition
evaluation(p31,success)

==============
Checking Rules
==============
Adding beliefs about the constructed plan
```

```
Applying rule12
User Model: Adding Proposition
bel(user,error(p1,p28))

================
Checking Rules
================
Adding goals

Applying rule8
User Model: Adding Proposition
goal(system,mb(system,user,replace(p1,_116192)))

==========
Responding
==========

Goal   : mb(system,user,replace(p1,_124779))
Effect: bel(user,goal(system,mb(system,user,replace(p1,_124779))))

---------
Construct
---------

Initial Setup
-------------
Trying: replace-plan(p1)
<con> error(p1,_128355)
<con> plan(user,p1,_128361)
<con> content(p1,p28,_128367)
New edge added
Trying: replace-plan(p1)
<con> error(p1,_128355)
<con> plan(user,p1,_128361)
<con> constraint(p1,_128366,p28)
<con> step(p1,_128372,p26)
<con> content(p1,p16,_128379)
<con> modifiers(entity(1,_145806),[antenna1,fern1])=modifier(_128386,_128387,_128388)
*** failed ***
Trying: expand-plan(p1)
<con> error(p1,_128355)
<con> plan(user,p1,_128361)
<con> constraint(p1,_128366,p28)
<con> content(p1,p26,_128373)
<con> yield(p1,p26,[])
<con> modifiers(entity(1,_141691),[antenna1,fern1])=modifiers(_128386,_128387)
New edge added

End of Initialization
---------------------

Expanding (step 0  active 2)
----------------------------
Head: expand-plan(p1)
Need: member(_128167,[antenna1,fern1])
<ma>  member(_128167,[antenna1,fern1])
New edge added

Expanding (step 1  active 2)
----------------------------
Head: replace-plan(p1)
Need: relax-constraint([antenna1,fern1]=[_128404],_128396)
No edges added

Expanding (step 2  active 1)
----------------------------
Head: expand-plan(p1)
Need: ref(entity(1,_133813),antenna1)
```

```
<ma>  ref(entity(1,_133813),antenna1)
New edge added

Expanding (step 3  active 1)
----------------------------
Head: expand-plan(p1)
Need: construct(modifiers(entity(1,antenna1),[antenna1,fern1]),_143912,_143913)

Invoking construct recursively

---------
Construct
---------

Initial Setup
-------------
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
<con> [antenna1,fern1]=[antenna1]
*** failed ***
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added

End of Initialization
---------------------

Expanding (step 0  active 2)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifier(entity(1,antenna1),[antenna1,fern1],_149322)
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_149322)
New edge added
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_149322)
New edge added

Expanding (step 1  active 3)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel-group(entity(1,antenna1),_149253,[antenna1,fern1])
Trying: s-attrib-rel-group(entity(1,antenna1),_165181^max(size,_165181),[antenna1,fern1])
<con> mb(system,user,max(size,antenna1,[antenna1,fern1]))
*** failed ***
No edges added

Expanding (step 2  active 2)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(1,antenna1),_157458)
Trying: s-attrib(entity(1,antenna1),_171563^category(_171563,_171567))
<con> mb(system,user,category(antenna1,_171567))
New edge added
Trying: s-attrib(entity(1,antenna1),_171563^colour(_171563,_171567))
<con> mb(system,user,colour(antenna1,_171567))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_171563^size(_171563,_171567))
<con> mb(system,user,size(antenna1,_171567))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_171563^assessment(_171563,_171567))
<con> mb(system,user,assessment(antenna1,_171567))
New edge added

Expanding (step 3  active 3)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel(entity(1,antenna1),_157300,_157301)
Trying: s-attrib-rel(entity(1,antenna1),_157300,_178834^_178837^at(_178834,_178837))
<con> mb(system,user,at(antenna1,_178859))
```

```
*** failed ***
Trying: s-attrib-rel(entity(1,antenna1),_157300,_178834^_178837^on(_178834,_178837))
<con> mb(system,user,on(antenna1,_178859))
New edge added
Trying: s-attrib-rel(entity(1,antenna1),_157300,_178834^_178837^in(_178834,_178837))
<con> mb(system,user,in(antenna1,_178859))
*** failed ***

Expanding (step 4  active 3)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_171405^ub(system,user,apply(_171415^assessment(_171415,weird),_171405))
       - ,_171399)
<ma>  subset([antenna1,fern1],_171405^ub(system,user,apply(_171415^assessment(_171415,weird),_171405))
       - ,_171399)
New edge added

Expanding (step 5  active 3)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_171592^ub(system,user,apply(_171602^category(_171602,creature),_171592)
       - ),_171586)
<ma>  subset([antenna1,fern1],_171592^ub(system,user,apply(_171602^category(_171602,creature),_171592)
       - ),_171586)
New edge added

Expanding (step 6  active 3)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: ref(entity(3,television1),_178668)
<ma>  ref(entity(3,television1),_178668)
New edge added

Expanding (step 7  active 3)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifiers(entity(1,antenna1),[antenna1,fern1])
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
<con> [antenna1,fern1]=[antenna1]
*** failed ***
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added

Expanding (step 8  active 4)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifiers(entity(1,antenna1),[antenna1,fern1])
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
<con> [antenna1,fern1]=[antenna1]
*** failed ***
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added

Expanding (step 9  active 5)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_207145^ub(system,user,apply(apply(_207158^_207161^on(_207158,_207161),
       - _207145),television1)),_207139)
<ma>  subset([antenna1,fern1],_207145^ub(system,user,apply(apply(_207158^_207161^on(_207158,_207161),
       - _207145),television1)),_207139)
New edge added

Expanding (step 10  active 5)
-----------------------------
```

```
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifier(entity(1,antenna1),[antenna1,fern1],_214782)
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_214782)
New edge added
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_214782)
New edge added

Expanding (step 11  active 6)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel-group(entity(1,antenna1),_214566,[antenna1,fern1])
Trying: s-attrib-rel-group(entity(1,antenna1),_251497^max(size,_251497),[antenna1,fern1])
<con> mb(system,user,max(size,antenna1,[antenna1,fern1]))
*** failed ***
No edges added

Expanding (step 12  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifier(entity(1,antenna1),[antenna1,fern1],_222591)
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_222591)
New edge added
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_222591)
New edge added

Expanding (step 13  active 6)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel-group(entity(1,antenna1),_222375,[antenna1,fern1])
Trying: s-attrib-rel-group(entity(1,antenna1),_267226^max(size,_267226),[antenna1,fern1])
<con> mb(system,user,max(size,antenna1,[antenna1,fern1]))
*** failed ***
No edges added

Expanding (step 14  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: refer(entity(3,television1))
Trying: refer(entity(3,television1))
New edge added

Expanding (step 15  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(1,antenna1),_243201)
Trying: s-attrib(entity(1,antenna1),_280236^category(_280236,_280240))
<con> mb(system,user,category(antenna1,_280240))
New edge added
Trying: s-attrib(entity(1,antenna1),_280236^colour(_280236,_280240))
<con> mb(system,user,colour(antenna1,_280240))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_280236^size(_280236,_280240))
<con> mb(system,user,size(antenna1,_280240))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_280236^assessment(_280236,_280240))
<con> mb(system,user,assessment(antenna1,_280240))
*** throwing out candidate (repeated action) ***

Expanding (step 16  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel(entity(1,antenna1),_242896,_242897)
Trying: s-attrib-rel(entity(1,antenna1),_242896,_287656^_287659^at(_287656,_287659))
<con> mb(system,user,at(antenna1,_287681))
*** failed ***
Trying: s-attrib-rel(entity(1,antenna1),_242896,_287656^_287659^on(_287656,_287659))
<con> mb(system,user,on(antenna1,_287681))
New edge added
```

```
Trying: s-attrib-rel(entity(1,antenna1),_242896,_287656^_287659^in(_287656,_287659))
<con> mb(system,user,in(antenna1,_287681))
*** failed ***

Expanding (step 17  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(1,antenna1),_258930)
Trying: s-attrib(entity(1,antenna1),_294768^category(_294768,_294772))
<con> mb(system,user,category(antenna1,_294772))
*** throwing out candidate (repeated action) ***
Trying: s-attrib(entity(1,antenna1),_294768^colour(_294768,_294772))
<con> mb(system,user,colour(antenna1,_294772))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_294768^size(_294768,_294772))
<con> mb(system,user,size(antenna1,_294772))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_294768^assessment(_294768,_294772))
<con> mb(system,user,assessment(antenna1,_294772))
New edge added

Expanding (step 18  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel(entity(1,antenna1),_258625,_258626)
Trying: s-attrib-rel(entity(1,antenna1),_258625,_302188^_302191^at(_302188,_302191))
<con> mb(system,user,at(antenna1,_302213))
*** failed ***
Trying: s-attrib-rel(entity(1,antenna1),_258625,_302188^_302191^on(_302188,_302191))
<con> mb(system,user,on(antenna1,_302213))
New edge added
Trying: s-attrib-rel(entity(1,antenna1),_258625,_302188^_302191^in(_302188,_302191))
<con> mb(system,user,in(antenna1,_302213))
*** failed ***

Expanding (step 19  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-refer(entity(3,television1))
Trying: s-refer(entity(3,television1))
New edge added

Expanding (step 20  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_280078^ub(system,user,apply(_280088^category(_280088,creature),_280078)
      - ),_280072)
<ma>  subset([antenna1,fern1],_280078^ub(system,user,apply(_280088^category(_280088,creature),_280078)
      - ),_280072)
New edge added

Expanding (step 21  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: ref(entity(6,television1),_287490)
<ma>  ref(entity(6,television1),_287490)
New edge added

Expanding (step 22  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_294610^ub(system,user,apply(_294620^assessment(_294620,weird),_294610))
      - ,_294604)
<ma>  subset([antenna1,fern1],_294610^ub(system,user,apply(_294620^assessment(_294620,weird),_294610))
      - ,_294604)
New edge added

Expanding (step 23  active 5)
```

```
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: ref(entity(9,television1),_302022)
<ma>  ref(entity(9,television1),_302022)
New edge added

Expanding (step 24  active 5)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: describe(entity(3,television1))
Trying: describe(entity(3,television1))
New edge added

Expanding (step 25  active 5)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifiers(entity(1,antenna1),[antenna1,fern1])
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
<con> [antenna1,fern1]=[antenna1]
*** failed ***
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added

Expanding (step 26  active 6)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_327200^ub(system,user,apply(apply(_327213^_327216^on(_327213,_327216),
      - _327200),television1)),_327194)
<ma>  subset([antenna1,fern1],_327200^ub(system,user,apply(apply(_327213^_327216^on(_327213,_327216),
      - _327200),television1)),_327194)
New edge added

Expanding (step 27  active 6)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifiers(entity(1,antenna1),[antenna1,fern1])
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
<con> [antenna1,fern1]=[antenna1]
*** failed ***
Trying: modifiers(entity(1,antenna1),[antenna1,fern1])
New edge added

Expanding (step 28  active 7)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_345841^ub(system,user,apply(apply(_345854^_345857^on(_345854,_345857),
      - _345841),television1)),_345835)
<ma>  subset([antenna1,fern1],_345841^ub(system,user,apply(apply(_345854^_345857^on(_345854,_345857),
      - _345841),television1)),_345835)
New edge added

Expanding (step 29  active 7)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: headnoun(entity(3,television1),_352654)
Trying: headnoun(entity(3,television1),_352654)
New edge added

Expanding (step 30  active 7)
----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifier(entity(1,antenna1),[antenna1,fern1],_360849)
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_360849)
New edge added
```

```
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_360849)
New edge added

Expanding (step 31  active 8)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel-group(entity(1,antenna1),_360486,[antenna1,fern1])
Trying: s-attrib-rel-group(entity(1,antenna1),_419797^max(size,_419797),[antenna1,fern1])
<con> mb(system,user,max(size,antenna1,[antenna1,fern1]))
*** failed ***
No edges added

Expanding (step 32  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: refer(entity(6,television1))
Trying: refer(entity(6,television1))
New edge added

Expanding (step 33  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifier(entity(1,antenna1),[antenna1,fern1],_381996)
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_381996)
New edge added
Trying: modifier(entity(1,antenna1),[antenna1,fern1],_381996)
New edge added

Expanding (step 34  active 8)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel-group(entity(1,antenna1),_381633,[antenna1,fern1])
Trying: s-attrib-rel-group(entity(1,antenna1),_443255^max(size,_443255),[antenna1,fern1])
<con> mb(system,user,max(size,antenna1,[antenna1,fern1]))
*** failed ***
No edges added

Expanding (step 35  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: refer(entity(9,television1))
Trying: refer(entity(9,television1))
New edge added

Expanding (step 36  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(3,television1),_402197^category(_402197,_402201))
Trying: s-attrib(entity(3,television1),_402197^category(_402197,_402201))
<con> mb(system,user,category(television1,_27361))
New edge added

Expanding (step 37  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(1,antenna1),_28237)
Trying: s-attrib(entity(1,antenna1),_38565^category(_38565,_38569))
<con> mb(system,user,category(antenna1,_38569))
*** throwing out candidate (repeated action) ***
Trying: s-attrib(entity(1,antenna1),_38565^colour(_38565,_38569))
<con> mb(system,user,colour(antenna1,_38569))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_38565^size(_38565,_38569))
<con> mb(system,user,size(antenna1,_38569))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_38565^assessment(_38565,_38569))
<con> mb(system,user,assessment(antenna1,_38569))
*** throwing out candidate (repeated action) ***
```

```
No edges added

Expanding (step 38  active 6)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel(entity(1,antenna1),_27799,_27800)
Trying: s-attrib-rel(entity(1,antenna1),_27799,_45207^_45210^at(_45207,_45210))
<con> mb(system,user,at(antenna1,_45232))
*** failed ***
Trying: s-attrib-rel(entity(1,antenna1),_27799,_45207^_45210^on(_45207,_45210))
<con> mb(system,user,on(antenna1,_45232))
New edge added
Trying: s-attrib-rel(entity(1,antenna1),_27799,_45207^_45210^in(_45207,_45210))
<con> mb(system,user,in(antenna1,_45232))
*** failed ***

Expanding (step 39  active 6)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-refer(entity(6,television1))
Trying: s-refer(entity(6,television1))
New edge added

Expanding (step 40  active 6)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(1,antenna1),_29642)
Trying: s-attrib(entity(1,antenna1),_59879^category(_59879,_59883))
<con> mb(system,user,category(antenna1,_59883))
*** throwing out candidate (repeated action) ***
Trying: s-attrib(entity(1,antenna1),_59879^colour(_59879,_59883))
<con> mb(system,user,colour(antenna1,_59883))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_59879^size(_59879,_59883))
<con> mb(system,user,size(antenna1,_59883))
*** failed ***
Trying: s-attrib(entity(1,antenna1),_59879^assessment(_59879,_59883))
<con> mb(system,user,assessment(antenna1,_59883))
*** throwing out candidate (repeated action) ***
No edges added

Expanding (step 41  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel(entity(1,antenna1),_29204,_29205)
Trying: s-attrib-rel(entity(1,antenna1),_29204,_66521^_66524^at(_66521,_66524))
<con> mb(system,user,at(antenna1,_66546))
*** failed ***
Trying: s-attrib-rel(entity(1,antenna1),_29204,_66521^_66524^on(_66521,_66524))
<con> mb(system,user,on(antenna1,_66546))
New edge added
Trying: s-attrib-rel(entity(1,antenna1),_29204,_66521^_66524^in(_66521,_66524))
<con> mb(system,user,in(antenna1,_66546))
*** failed ***

Expanding (step 42  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-refer(entity(9,television1))
Trying: s-refer(entity(9,television1))
New edge added

Expanding (step 43  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([fern1,antenna1,corner1,television1],_30932^ub(system,user,category(_30932,television)),
       - _30922)
<ma>  subset([fern1,antenna1,corner1,television1],_30932^ub(system,user,category(_30932,television)),
```

```
       - _30922)
New edge added

Expanding (step 44  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: ref(entity(12,television1),_45041)
<ma>  ref(entity(12,television1),_45041)
New edge added

Expanding (step 45  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: describe(entity(6,television1))
Trying: describe(entity(6,television1))
New edge added

Expanding (step 46  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: ref(entity(15,television1),_66355)
<ma>  ref(entity(15,television1),_66355)
New edge added

Expanding (step 47  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: describe(entity(9,television1))
Trying: describe(entity(9,television1))
New edge added

Expanding (step 48  active 5)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifiers(entity(3,television1),[television1])
Trying: modifiers(entity(3,television1),[television1])
New edge added
Trying: modifiers(entity(3,television1),[television1])
<con> [television1]=[television1]
New edge added
Trying: modifiers(entity(3,television1),[television1])
New edge added

Expanding (step 49  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_92484^ub(system,user,apply(apply(_92497^_92500^on(_92497,_92500),
       - _92484),television1)),_92478)
<ma>  subset([antenna1,fern1],_92484^ub(system,user,apply(apply(_92497^_92500^on(_92497,_92500),
       - _92484),television1)),_92478)
New edge added

Expanding (step 50  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: headnoun(entity(6,television1),_99678)
Trying: headnoun(entity(6,television1),_99678)
New edge added

Expanding (step 51  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: subset([antenna1,fern1],_107298^ub(system,user,apply(apply(_107311^_107314^on(_107311,_107314),
       - _107298),television1)),_107292)
<ma>  subset([antenna1,fern1],_107298^ub(system,user,apply(apply(_107311^_107314^on(_107311,_107314),
       - _107298),television1)),_107292)
New edge added
```

```
Expanding (step 52  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: headnoun(entity(9,television1),_114492)
Trying: headnoun(entity(9,television1),_114492)
New edge added

Expanding (step 53  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: null
New edge added

Expanding (step 54  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifier(entity(3,television1),[television1],_123290)
Trying: modifier(entity(3,television1),[television1],_123290)
New edge added
Trying: modifier(entity(3,television1),[television1],_123290)
New edge added

Expanding (step 55  active 8)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel-group(entity(3,television1),_122327,[television1])
Trying: s-attrib-rel-group(entity(3,television1),_188455^max(size,_188455),[television1])
<con> mb(system,user,max(size,television1,[television1]))
*** failed ***
No edges added

Expanding (step 56  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: refer(entity(12,television1))
Trying: refer(entity(12,television1))
New edge added

Expanding (step 57  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(6,television1),_144010^category(_144010,_144014))
Trying: s-attrib(entity(6,television1),_144010^category(_144010,_144014))
<con> mb(system,user,category(television1,_144014))
New edge added

Expanding (step 58  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: refer(entity(15,television1))
Trying: refer(entity(15,television1))
New edge added

Expanding (step 59  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(9,television1),_164771^category(_164771,_164775))
Trying: s-attrib(entity(9,television1),_164771^category(_164771,_164775))
<con> mb(system,user,category(television1,_164775))
New edge added

Expanding (step 60  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: modifiers(entity(1,antenna1),[antenna1])
Trying: modifiers(entity(1,antenna1),[antenna1])
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1])
```

```
<con> [antenna1]=[antenna1]
New edge added
Trying: modifiers(entity(1,antenna1),[antenna1])
New edge added

Expanding (step 61  active 9)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib(entity(3,television1),_179685)
Trying: s-attrib(entity(3,television1),_237520^category(_237520,_237524))
<con> mb(system,user,category(television1,_237524))
*** throwing out candidate (repeated action) ***
Trying: s-attrib(entity(3,television1),_237520^colour(_237520,_237524))
<con> mb(system,user,colour(television1,_237524))
*** failed ***
Trying: s-attrib(entity(3,television1),_237520^size(_237520,_237524))
<con> mb(system,user,size(television1,_237524))
*** failed ***
Trying: s-attrib(entity(3,television1),_237520^assessment(_237520,_237524))
<con> mb(system,user,assessment(television1,_237524))
*** failed ***
No edges added

Expanding (step 62  active 8)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-attrib-rel(entity(3,television1),_179120,_179121)
Trying: s-attrib-rel(entity(3,television1),_179120,_244162^_244165^at(_244162,_244165))
<con> mb(system,user,at(television1,_244187))
*** failed ***
Trying: s-attrib-rel(entity(3,television1),_179120,_244162^_244165^on(_244162,_244165))
<con> mb(system,user,on(television1,_244187))
*** failed ***
Trying: s-attrib-rel(entity(3,television1),_179120,_244162^_244165^in(_244162,_244165))
<con> mb(system,user,in(television1,_244187))
*** failed ***
No edges added

Expanding (step 63  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-refer(entity(12,television1))
Trying: s-refer(entity(12,television1))
New edge added

Expanding (step 64  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: s-refer(entity(15,television1))
Trying: s-refer(entity(15,television1))
New edge added

Expanding (step 65  active 7)
-----------------------------
Head: modifiers(entity(1,antenna1),[antenna1,fern1])
Need: null
New edge added

Returning from recursive call to construct

Expanding (step 4  active 1)
----------------------------
Head: expand-plan(p1)
Need: substitute(p1,p26,V1,_16401,V2)
      V1=modifiers(entity(1,antenna1),[antenna1,fern1])
         +-modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
         | +-s-attrib-rel(entity(1,antenna1),entity(3,television1),_16476^_16479^on(_16476,_16479))
         | +-ref(entity(3,television1),television1)
```

```
        | +-subset([antenna1,fern1],_16536^ub(system,user,apply(apply(_16476^_16479^on(_16476,_16479)
        | |  - ,_16536),television1)),[antenna1])
        | +-refer(entity(3,television1))
        |   +-s-refer(entity(3,television1))
        |   +-describe(entity(3,television1))
        |     +-headnoun(entity(3,television1),[television1])
        |     | +-s-attrib(entity(3,television1),_16672^category(_16672,television))
        |     | +-subset([fern1,antenna1,corner1,television1],_16672^ub(system,user,category(_16672,
        |     |    - television)),[television1])
        |     +-modifiers(entity(3,television1),[television1])
        |       +-null
        +-modifiers(entity(1,antenna1),[antenna1])
          +-null
      V2=s-attrib(entity(3,television1),_16672^category(_16672,television))
        s-refer(entity(3,television1))
        s-attrib-rel(entity(1,antenna1),entity(3,television1),_16476^_16479^on(_16476,_16479))
<ma>  substitute(p1,p26,V1,_16401,V2)
      V1=modifiers(entity(1,antenna1),[antenna1,fern1])
        +-modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
        | +-s-attrib-rel(entity(1,antenna1),entity(3,television1),_16476^_16479^on(_16476,_16479))
        | +-ref(entity(3,television1),television1)
        | +-subset([antenna1,fern1],_16536^ub(system,user,apply(apply(_16476^_16479^on(_16476,_16479)
        | |  - ,_16536),television1)),[antenna1])
        | +-refer(entity(3,television1))
        |   +-s-refer(entity(3,television1))
        |   +-describe(entity(3,television1))
        |     +-headnoun(entity(3,television1),[television1])
        |     | +-s-attrib(entity(3,television1),_16672^category(_16672,television))
        |     | +-subset([fern1,antenna1,corner1,television1],_16672^ub(system,user,category(_16672,
        |     |    - television)),[television1])
        |     +-modifiers(entity(3,television1),[television1])
        |       +-null
        +-modifiers(entity(1,antenna1),[antenna1])
          +-null
      V2=s-attrib(entity(3,television1),_16672^category(_16672,television))
        s-refer(entity(3,television1))
        s-attrib-rel(entity(1,antenna1),entity(3,television1),_16476^_16479^on(_16476,_16479))
New edge added

Expanding (step 5  active 1)
----------------------------
Head: expand-plan(p1)
Need: s-reject(p1,V1)
      V1=s-attrib(entity(3,television1),_93809^category(_93809,television))
        s-refer(entity(3,television1))
        s-attrib-rel(entity(1,antenna1),entity(3,television1),_93834^_93837^on(_93834,_93837))
Trying: s-reject(p1,V1)
        V1=s-attrib(entity(3,television1),_93809^category(_93809,television))
          s-refer(entity(3,television1))
          s-attrib-rel(entity(1,antenna1),entity(3,television1),_93834^_93837^on(_93834,_93837))
New edge added


Constructed Plan
----------------
expand-plan(p1)
+-member(antenna1,[antenna1,fern1])
+-ref(entity(1,antenna1),antenna1)
+-construct(modifiers(entity(1,antenna1),[antenna1,fern1]),V1,V2)
| V1=modifiers(entity(1,antenna1),[antenna1,fern1])
|    +-modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
|    | +-s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,_118924))
|    | +-ref(entity(3,television1),television1)
|    | +-subset([antenna1,fern1],_119184^ub(system,user,apply(apply(_118921^_118924^on(_118921,_118924
|    | |  - ),_119184),television1)),[antenna1])
|    | +-refer(entity(3,television1))
|    |   +-s-refer(entity(3,television1))
|    |   +-describe(entity(3,television1))
```

```
|     |     +-headnoun(entity(3,television1),[television1])
|     |     | +-s-attrib(entity(3,television1),_118896^category(_118896,television))
|     |     | +-subset([fern1,antenna1,corner1,television1],_118896^ub(system,user,category(_118896,
|     |     |    - television)),[television1])
|     |     +-modifiers(entity(3,television1),[television1])
|     |       +-null
|     +-modifiers(entity(1,antenna1),[antenna1])
|       +-null
| V2=s-attrib(entity(3,television1),_118896^category(_118896,television))
|    s-refer(entity(3,television1))
|    s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,_118924))
+-substitute(p1,p26,V1,V2,V3)
| V1=modifiers(entity(1,antenna1),[antenna1,fern1])
|    +-modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
|    | +-s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,_118924))
|    | +-ref(entity(3,television1),television1)
|    | +-subset([antenna1,fern1],_119184^ub(system,user,apply(apply(_118921^_118924^on(_118921,_118924
|    | |  - ),_119184),television1)),[antenna1])
|    | +-refer(entity(3,television1))
|    |   +-s-refer(entity(3,television1))
|    |   +-describe(entity(3,television1))
|    |     +-headnoun(entity(3,television1),[television1])
|    |     | +-s-attrib(entity(3,television1),_118896^category(_118896,television))
|    |     | +-subset([fern1,antenna1,corner1,television1],_118896^ub(system,user,category(_118896,
|    |     |    - television)),[television1])
|    |     +-modifiers(entity(3,television1),[television1])
|    |       +-null
|    +-modifiers(entity(1,antenna1),[antenna1])
|      +-null
| V2=_119932:refer(entity(1,antenna1))
|    +-_119965:s-refer(entity(1,antenna1))
|    +-_119984:describe(entity(1,antenna1))
|      +-_120003:headnoun(entity(1,antenna1),_120016)
|      | +-_120044:s-attrib(entity(1,antenna1),_120062^category(_120062,creature))
|      | +-_120107:subset(_120040,_120062^ub(_120026,_120033,category(_120062,creature)),_120016)
|      +-_120126:modifiers(entity(1,antenna1),_120016)
|        +-_120146:modifier(entity(1,antenna1),_120016,[antenna1,fern1])
|        | +-_120185:s-attrib(entity(1,antenna1),_120203^assessment(_120203,weird))
|        | +-_120248:subset(_120016,_120255^ub(_120174,_120181,apply(_120203^assessment(_120203,weird)
|        |    - ,_120255)),[antenna1,fern1])
|        +-_120277:modifiers(entity(1,antenna1),[antenna1,fern1])
|          +-_120301:modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
|          | +-_120342:s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(
|          | |  - _118921,_118924))
|          | +-_120423:ref(entity(3,television1),television1)
|          | +-_120434:subset([antenna1,fern1],_119184^ub(system,user,apply(apply(_118921^_118924^on(
|          | |  - _118921,_118924),_119184),television1)),[antenna1])
|          | +-_120471:refer(entity(3,television1))
|          |   +-_120504:s-refer(entity(3,television1))
|          |   +-_120523:describe(entity(3,television1))
|          |     +-_120542:headnoun(entity(3,television1),[television1])
|          |     | +-_120593:s-attrib(entity(3,television1),_118896^category(_118896,television))
|          |     | +-_120656:subset([fern1,antenna1,corner1,television1],_118896^ub(system,user,
|          |     |    - category(_118896,television)),[television1])
|          |     +-_120685:modifiers(entity(3,television1),[television1])
|          |       +-_120730:null
|          +-_120752:modifiers(entity(1,antenna1),[antenna1])
|            +-_120797:null
| V3=s-attrib(entity(3,television1),_118896^category(_118896,television))
|    s-refer(entity(3,television1))
|    s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,_118924))
+-s-reject(p1,V1)
  V1=s-attrib(entity(3,television1),_118896^category(_118896,television))
    s-refer(entity(3,television1))
    s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,_118924))

Actions to be fed to generator
------------------------------
```

```
s-reject(p1,V1)
V1=s-attrib(entity(3,television1),_118896^category(_118896,television))
   s-refer(entity(3,television1))
   s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,_118924))

User Model: Adding Proposition
plan(system,p42,mb(system,user,replace(p1,V1)))
V1=_119932:refer(entity(1,antenna1))
   +-_119965:s-refer(entity(1,antenna1))
   +-_119984:describe(entity(1,antenna1))
     +-_120003:headnoun(entity(1,antenna1),_120016)
     | +-_120044:s-attrib(entity(1,antenna1),_120062^category(_120062,creature))
     | +-_120107:subset(_120040,_120062^ub(_120026,_120033,category(_120062,creature)),_120016)
     +-_120126:modifiers(entity(1,antenna1),_120016)
       +-_120146:modifier(entity(1,antenna1),_120016,[antenna1,fern1])
       | +-_120185:s-attrib(entity(1,antenna1),_120203^assessment(_120203,weird))
       | +-_120248:subset(_120016,_120255^ub(_120174,_120181,apply(_120203^assessment(_120203,weird),
       |    - _120255)),[antenna1,fern1])
       +-_120277:modifiers(entity(1,antenna1),[antenna1,fern1])
         +-_120301:modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
         | +-_120342:s-attrib-rel(entity(1,antenna1),entity(3,television1),_118921^_118924^on(_118921,
         | |  - _118924))
         | +-_120423:ref(entity(3,television1),television1)
         | +-_120434:subset([antenna1,fern1],_119184^ub(system,user,apply(apply(_118921^_118924^on(
         | |  - _118921,_118924),_119184),television1)),[antenna1])
         | +-_16546:refer(entity(3,television1))
         |   +-_16579:s-refer(entity(3,television1))
         |   +-_16598:describe(entity(3,television1))
         |     +-_16617:headnoun(entity(3,television1),[television1])
         |     | +-_16668:s-attrib(entity(3,television1),_14971^category(_14971,television))
         |     | +-_16731:subset([fern1,antenna1,corner1,television1],_14971^ub(system,user,category(
         |     |    - _14971,television)),[television1])
         |     +-_16760:modifiers(entity(3,television1),[television1])
         |       +-_16805:null
         +-_16827:modifiers(entity(1,antenna1),[antenna1])
           +-_16872:null

User Model: Adding Proposition
contributed(system,p42)

User Model: Adding Proposition
evaluation(p42,success)

==============
Checking Rules
==============
Adding beliefs about the constructed plan

Applying rule11
User Model: Adding Proposition
replace(p1,V1)
V1=_78726:refer(entity(1,antenna1))
   +-_78759:s-refer(entity(1,antenna1))
   +-_78778:describe(entity(1,antenna1))
     +-_78797:headnoun(entity(1,antenna1),_78810)
     | +-_78838:s-attrib(entity(1,antenna1),_78856^category(_78856,creature))
     | +-_78901:subset(_78834,_78856^ub(_78820,_78827,category(_78856,creature)),_78810)
     +-_78920:modifiers(entity(1,antenna1),_78810)
       +-_78940:modifier(entity(1,antenna1),_78810,[antenna1,fern1])
       | +-_78979:s-attrib(entity(1,antenna1),_78997^assessment(_78997,weird))
       | +-_79042:subset(_78810,_79049^ub(_78968,_78975,apply(_78997^assessment(_78997,weird),_79049))
       |    - ,[antenna1,fern1])
       +-_79071:modifiers(entity(1,antenna1),[antenna1,fern1])
         +-_79095:modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])
         | +-_79136:s-attrib-rel(entity(1,antenna1),entity(3,television1),_79158^_79161^on(_79158,
         | |  - _79161))
         | +-_79217:ref(entity(3,television1),television1)
         | +-_79228:subset([antenna1,fern1],_79239^ub(system,user,apply(apply(_79158^_79161^on(_79158,
```

```
| |  - _79161),_79239),television1)),[antenna1])
| +-_79265:refer(entity(3,television1))
|   +-_79298:s-refer(entity(3,television1))
|   +-_79317:describe(entity(3,television1))
|     +-_79336:headnoun(entity(3,television1),[television1])
|     | +-_79387:s-attrib(entity(3,television1),_79405^category(_79405,television))
|     | +-_79450:subset([fern1,antenna1,corner1,television1],_79405^ub(system,user,category(
|     |     - _79405,television)),[television1])
|     +-_79479:modifiers(entity(3,television1),[television1])
|       +-_79524:null
+-_79546:modifiers(entity(1,antenna1),[antenna1])
  +-_79591:null

removing plan p1 from belief space

User Model: Adding Proposition
contributed(system,p57)

User Model: Adding Proposition
evaluation(p57,success)

User Model: Adding Proposition
mgoal(system,user,p57,knowref(system,entity(1,_205950)))

*** mgoal updated ***

Applying rule13
User Model: Adding Proposition
achieve(p57,knowref(system,entity(1,antenna1)))

==========
Checking Rules
==========
Adding goals
```

A.4 Understanding "No, on the television"

```
==========
Understanding
==========

----------
Plan Recognizer
----------

Actions = s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_238266^_238269^on(_238266,_238269)
              - )

Initial Setup
----------
Adding inactive edge:
Head: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_252974^_252977^on(_252974,_252977))
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_252974^_252977^on(_252974,_252977))

Adding active edge:
Head: reject-plan(_267622)
Need: s-reject(_267622,_267664)

Adding config (from active edge):
Active:   reject-plan(_267622)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_252974^_252977^on(_252974,_252977)
              - )
```

```
End of Initial Setup
--------------------

Applying config (0 left, step 1)
--------------------------------
Active:   reject-plan(_267622)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_252974^_252977^on(_252974,_252977)
              - )

Adding inactive edge:
Head: reject-plan(p57)
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_287902^_287905^on(_287902,_287905))

*** Agenda Empty ***

Number of complete plans found: 1

-----------------------
Consider Plan Derivation
-----------------------

reject-plan(p57)
+-s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_298094^_298097^on(_298094,_298097))

--------
Evaluate
--------

<con> error(p57,_298062)
<con> plan(system,p57,_298068)
postponing evaluation of constraint(p57,_298073,_298062)
<con> yield(p57,_298073,[s-attrib-rel(entity(1,antenna1),entity(3,television1),_298094^_298097^on(
       - _298094,_298097))])
<con> constraint(p57,p86,_298062)
<con> length([s-attrib-rel(entity(1,antenna1),entity(3,television1),_298094^_298097^on(_298094,_298097
       - ))],1)


Inferred Plan
-------------

_306665:reject-plan(p57)
+-_307496:s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,antenna1),entity(3,television1),_298094^_298097^on(_298094,_298097))

User Model: Adding Proposition
plan(user,p119,mb(system,user,error(p57,p91)))

User Model: Adding Proposition
contributed(user,p119)

User Model: Adding Proposition
evaluation(p119,success)

==============
Checking Rules
==============
Adding beliefs about the inferred plan

Applying rule2
User Model: Adding Proposition
bel(user,error(p57,p91))

Applying rule3
```

```
User Model: Adding Proposition
error(p57,p91)

==========
Understanding
==========

----------
Plan Recognizer
----------

Actions = s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_383536),entity(20,_383561),_383583^_383586^on(_383583,_383586))
             s-refer(entity(20,_383561))
             s-attrib(entity(20,_22021),_22061^category(_22061,television))

Initial Setup
----------
Adding inactive edge:
Head: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
         s-refer(entity(20,_66485))
         s-attrib(entity(20,_66485),_66511^category(_66511,television))
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
         s-refer(entity(20,_66485))
         s-attrib(entity(20,_66485),_66511^category(_66511,television))

Adding active edge:
Head: replace-plan(_90143)
Need: relax-constraint(_90179,_90187)

***Topdown addition of inactive entries (2)***

Adding inactive edge:
Head: relax-constraint(mb(_93359,_93360,category(_93363,_93364)),mb(_93359,_93360,_93368))

Adding config (from inactive edge):
Active:   replace-plan(_90143)
Inactive: relax-constraint(mb(_93359,_93360,category(_93363,_93364)),mb(_93359,_93360,_93368))

Adding active edge:
Head: replace-plan(_98155)
Need: relax-constraint(_98191,_98199)

***duplicate active edge (removed)***

Adding inactive edge:
Head: relax-constraint(mb(_93263,_93264,colour(_93267,_93268)),mb(_93263,_93264,_93272))

Adding config (from inactive edge):
Active:   replace-plan(_90143)
Inactive: relax-constraint(mb(_93263,_93264,colour(_93267,_93268)),mb(_93263,_93264,_93272))

Adding active edge:
Head: replace-plan(_121095)
Need: relax-constraint(_121131,_121139)

***duplicate active edge (removed)***

Adding active edge:
Head: replace-plan(_90008)
Need: s-reject(_90008,_90081)

Adding config (from active edge):
Active:   replace-plan(_90008)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
```

```
            s-refer(entity(20,_66485))
            s-attrib(entity(20,_66485),_66511^category(_66511,television))

Adding active edge:
Head: expand-plan(_89879)
Need: s-reject(_89879,_89951)

Adding config (from active edge):
Active:   expand-plan(_89879)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

End of Initial Setup
--------------------

Applying config (3 left, step 1)
---------------------------------
Active:   replace-plan(_90143)
Inactive: relax-constraint(mb(_93359,_93360,category(_93363,_93364)),mb(_93359,_93360,_93368))

Adding active edge:
Head: replace-plan(_176055)
Need: s-reject(_176055,_176101)

Adding config (from active edge):
Active:   replace-plan(_176055)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

Applying config (3 left, step 2)
---------------------------------
Active:   replace-plan(_90143)
Inactive: relax-constraint(mb(_93263,_93264,colour(_93267,_93268)),mb(_93263,_93264,_93272))

Adding active edge:
Head: replace-plan(_197101)
Need: s-reject(_197101,_197147)

Adding config (from active edge):
Active:   replace-plan(_197101)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

Applying config (3 left, step 3)
---------------------------------
Active:   replace-plan(_90008)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

Adding inactive edge:
Head: replace-plan(p57)
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,_226201),entity(20,_226204),_226206^_226209^on(_226206,_226209))
         s-refer(entity(20,_226204))
         s-attrib(entity(20,_226204),_226230^category(_226230,television))

Applying config (2 left, step 4)
---------------------------------
Active:   expand-plan(_89879)
Inactive: s-reject(p57,V1)
```

```
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

Adding inactive edge:
Head: expand-plan(p57)
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,_252925),entity(20,_252928),_252930^_252933^on(_252930,_252933))
         s-refer(entity(20,_252928))
         s-attrib(entity(20,_252928),_252954^category(_252954,television))

Applying config (1 left, step 5)
--------------------------------
Active:   replace-plan(_176055)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

Adding inactive edge:
Head: replace-plan(p57)
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,_279611),entity(20,_279614),_279616^_279619^on(_279616,_279619))
         s-refer(entity(20,_279614))
         s-attrib(entity(20,_279614),_279640^category(_279640,television))

Applying config (0 left, step 6)
--------------------------------
Active:   replace-plan(_197101)
Inactive: s-reject(p57,V1)
          V1=s-attrib-rel(entity(1,_66482),entity(20,_66485),_66487^_66490^on(_66487,_66490))
             s-refer(entity(20,_66485))
             s-attrib(entity(20,_66485),_66511^category(_66511,television))

Adding inactive edge:
Head: replace-plan(p57)
Acts: s-reject(p57,V1)
      V1=s-attrib-rel(entity(1,_306218),entity(20,_306221),_306223^_306226^on(_306223,_306226))
         s-refer(entity(20,_306221))
         s-attrib(entity(20,_306221),_306247^category(_306247,television))

*** Agenda Empty ***

Number of complete plans found: 4

----------------------
Consider Plan Derivation
----------------------

replace-plan(p57)
+-relax-constraint(mb(_323701,_323702,colour(_323705,_323706)),mb(_323701,_323702,_323731))
| +-_323731=colour(_323705,_323745)
+-substitute(p57,_323687,V1,_323778,V2)
| V1=mb(_323701,_323702,_323731)
| V2=s-attrib-rel(entity(1,_323792),entity(20,_323795),_323797^_323800^on(_323797,_323800))
|    s-refer(entity(20,_323795))
|    s-attrib(entity(20,_323795),_323821^category(_323821,television))
+-evaluate(_323778)
+-s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,_323792),entity(20,_323795),_323797^_323800^on(_323797,_323800))
     s-refer(entity(20,_323795))
     s-attrib(entity(20,_323795),_323821^category(_323821,television))

--------
Evaluate
--------

<con> error(p57,_323687)
```

```
<con> plan(system,p57,_323693)
<con> content(p57,p91,mb(system,user,colour(_323705,_323706)))
 *** failed ***

Failure Evaluation
------------------
Failure Node: content(p57,p91,mb(system,user,colour(_323705,_323706)))


------------------------
Consider Plan Derivation
------------------------

replace-plan(p57)
+-relax-constraint(mb(_370560,_370561,category(_370564,_370565)),mb(_370560,_370561,_370590))
| +-_370590=category(_370564,_370604)
+-substitute(p57,_370546,V1,_370637,V2)
| V1=mb(_370560,_370561,_370590)
| V2=s-attrib-rel(entity(1,_370651),entity(20,_370654),_370656^_370659^on(_370656,_370659))
|    s-refer(entity(20,_370654))
|    s-attrib(entity(20,_370654),_370680^category(_370680,television))
+-evaluate(_370637)
+-s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,_370651),entity(20,_370654),_370656^_370659^on(_370656,_370659))
     s-refer(entity(20,_370654))
     s-attrib(entity(20,_370654),_370680^category(_370680,television))

--------
Evaluate
--------

<con> error(p57,_370546)
<con> plan(system,p57,_370552)
<con> content(p57,p91,mb(system,user,category(_370564,_370565)))
 *** failed ***

Failure Evaluation
------------------
Failure Node: content(p57,p91,mb(system,user,category(_370564,_370565)))


------------------------
Consider Plan Derivation
------------------------

expand-plan(p57)
+-member(_417920,_417916)
+-ref(_417915,_417920)
+-construct(modifiers(_417915,_417916),_417931,V1)
| V1=s-attrib-rel(entity(1,_417944),entity(20,_417947),_417949^_417952^on(_417949,_417952))
|    s-refer(entity(20,_417947))
|    s-attrib(entity(20,_417947),_417973^category(_417973,television))
+-substitute(p57,_417895,_417931,_417984,V1)
| V1=s-attrib-rel(entity(1,_417944),entity(20,_417947),_417949^_417952^on(_417949,_417952))
|    s-refer(entity(20,_417947))
|    s-attrib(entity(20,_417947),_417973^category(_417973,television))
+-s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,_417944),entity(20,_417947),_417949^_417952^on(_417949,_417952))
     s-refer(entity(20,_417947))
     s-attrib(entity(20,_417947),_417973^category(_417973,television))

--------
Evaluate
--------

<con> error(p57,_66677)
<con> plan(system,p57,_66683)
<con> constraint(p57,_66688,p91)
```

```
<con> content(p57,p86,_66695)
<con> yield(p57,p86,[])
 *** failed ***

Failure Evaluation
------------------
Failure Mode: yield(p57,p86,[])


------------------------
Consider Plan Derivation
------------------------

replace-plan(p57)
+-member(_79024,_79019)
+-ref(_79018,_79024)
+-construct(modifier(_79018,_79019,_79020),_79035,V1)
| V1=s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))
|    s-refer(entity(20,_79052))
|    s-attrib(entity(20,_79052),_79078^category(_79078,television))
+-substitute(p57,_79004,_79035,_79089,V1)
| V1=s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))
|    s-refer(entity(20,_79052))
|    s-attrib(entity(20,_79052),_79078^category(_79078,television))
+-evaluate(_79089)
+-s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))
     s-refer(entity(20,_79052))
     s-attrib(entity(20,_79052),_79078^category(_79078,television))

--------
Evaluate
--------

<con> error(p57,_78987)
<con> plan(system,p57,_78993)
<con> constraint(p57,_78998,p91)
<con> step(p57,_79004,p86)
<con> content(p57,p83,_79011)
<con> modifier(entity(1,antenna1),[antenna1,fern1],[antenna1])=modifier(_79018,_79019,_79020)
postponing evaluation of member(_79024,[antenna1,fern1])
<ma>  ref(entity(1,antenna1),_79024)
<ma>  member(antenna1,[antenna1,fern1])
<ma>  construct(modifier(entity(1,antenna1),[antenna1,fern1],[antenna1]),_79035,V1)
      V1=s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))
         s-refer(entity(20,_79052))
         s-attrib(entity(20,_79052),_79078^category(_79078,television))

Invoking plan recognizer from evaluate

---------------
Plan Recognizer
---------------

Actions = s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))
          s-refer(entity(20,_79052))
          s-attrib(entity(20,_79052),_79078^category(_79078,television))

Initial Setup
-------------
Adding inactive edge:
Head: s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))
Acts: s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))

Adding active edge:
Head: modifier(_203003,_203004,_203005)
Need: s-attrib-rel(_203003,_203032,_203051)
```

```
Adding config (from active edge):
Active:   modifier(_203003,_203004,_203005)
Inactive: s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))

Adding inactive edge:
Head: s-refer(entity(20,_79052))
Acts: s-refer(entity(20,_79052))

Adding active edge:
Head: refer(_218166)
Need: s-refer(_218166)

Adding config (from active edge):
Active:   refer(_218166)
Inactive: s-refer(entity(20,_79052))

Adding inactive edge:
Head: s-attrib(entity(20,_79052),_79078^category(_79078,television))
Acts: s-attrib(entity(20,_79052),_79078^category(_79078,television))

Adding active edge:
Head: headnoun(_231526,_231527)
Need: s-attrib(_231526,_231561^category(_231561,_231569))

Adding config (from active edge):
Active:   headnoun(_231526,_231527)
Inactive: s-attrib(entity(20,_79052),_79078^category(_79078,television))

Adding active edge:
Head: modifier(_231456,_231457,_231458)
Need: s-attrib(_231456,_231496)

Adding config (from active edge):
Active:   modifier(_231456,_231457,_231458)
Inactive: s-attrib(entity(20,_79052),_79078^category(_79078,television))

End of Initial Setup
--------------------

Applying config (3 left, step 1)
--------------------------------
Active:   modifier(_203003,_203004,_203005)
Inactive: s-attrib-rel(entity(1,_79049),entity(20,_79052),_79054^_79057^on(_79054,_79057))

Adding active edge:
Head: modifier(entity(1,_258735),_258550,_258551)
Need: refer(entity(20,_258738))
Acts: s-attrib-rel(entity(1,_258735),entity(20,_258738),_258740^_258743^on(_258740,_258743))

Applying config (2 left, step 2)
--------------------------------
Active:   refer(_218166)
Inactive: s-refer(entity(20,_79052))

Adding active edge:
Head: refer(entity(20,_272825))
Need: describe(entity(20,_272825))
Acts: s-refer(entity(20,_272825))

Applying config (1 left, step 3)
--------------------------------
Active:   headnoun(_231526,_231527)
Inactive: s-attrib(entity(20,_79052),_79078^category(_79078,television))

Adding inactive edge:
Head: headnoun(entity(20,_284967),_284793)
Acts: s-attrib(entity(20,_284967),_284827^category(_284827,television))
```

```
Adding active edge:
Head: describe(_291203)
Need: headnoun(_291203,_291226)

Adding config (from active edge):
Active:   describe(_291203)
Inactive: headnoun(entity(20,_284967),_284793)

Applying config (1 left, step 4)
--------------------------------
Active:   modifier(_231456,_231457,_231458)
Inactive: s-attrib(entity(20,_79052),_79078^category(_79078,television))

Adding inactive edge:
Head: modifier(entity(20,_304378),_304212,_304213)
Acts: s-attrib(entity(20,_304378),_304380^category(_304380,television))

Adding active edge:
Head: modifiers(_310937,_310938)
Need: modifier(_310937,_310938,_310963)

Adding config (from active edge):
Active:   modifiers(_310937,_310938)
Inactive: modifier(entity(20,_304378),_304212,_304213)

Applying config (1 left, step 5)
--------------------------------
Active:   describe(_291203)
Inactive: headnoun(entity(20,_284967),_284793)

Adding active edge:
Head: describe(entity(20,_323642))
Need: modifiers(entity(20,_323642),_323516)
Acts: s-attrib(entity(20,_323642),_323688^category(_323688,television))

***Topdown addition of inactive entries (1)***

Adding inactive edge:
Head: modifiers(entity(20,_331732),_331729)

Adding config (from inactive edge):
Active:   describe(entity(20,_323642))
Inactive: modifiers(entity(20,_331732),_331729)

Applying config (1 left, step 6)
--------------------------------
Active:   modifiers(_310937,_310938)
Inactive: modifier(entity(20,_304378),_304212,_304213)

Adding active edge:
Head: modifiers(entity(20,_345018),_344866)
Need: modifiers(entity(20,_345018),_344891)
Acts: s-attrib(entity(20,_345018),_345061^category(_345061,television))

Adding config (from active edge):
Active:   modifiers(entity(20,_345018),_344866)
Inactive: modifiers(entity(20,_331732),_331729)

***Topdown addition of inactive entries (1)***

Adding inactive edge:
Head: modifiers(entity(20,_353432),_353429)

***duplicate inactive edge***

Applying config (1 left, step 7)
--------------------------------
Active:   describe(entity(20,_323642))
```

```
Inactive: modifiers(entity(20,_331732),_331729)

Adding inactive edge:
Head: describe(entity(20,_372089))
Acts: s-attrib(entity(20,_372089),_372151^category(_372151,television))

Adding config (from inactive edge):
Active:   refer(entity(20,_272825))
Inactive: describe(entity(20,_372089))

Applying config (1 left, step 8)
--------------------------------
Active:   modifiers(entity(20,_345018),_344866)
Inactive: modifiers(entity(20,_331732),_331729)

Adding inactive edge:
Head: modifiers(entity(20,_387952),_387949)
Acts: s-attrib(entity(20,_387952),_388012^category(_388012,television))

Applying config (0 left, step 9)
--------------------------------
Active:   refer(entity(20,_272825))
Inactive: describe(entity(20,_372089))

Adding inactive edge:
Head: refer(entity(20,_399351))
Acts: s-refer(entity(20,_399351))
      s-attrib(entity(20,_399351),_399601^category(_399601,television))

Adding config (from inactive edge):
Active:   modifier(entity(1,_258735),_258550,_258551)
Inactive: refer(entity(20,_399351))

Applying config (0 left, step 10)
---------------------------------
Active:   modifier(entity(1,_258735),_258550,_258551)
Inactive: refer(entity(20,_399351))

Adding inactive edge:
Head: modifier(entity(1,_417538),_417534,_417535)
Acts: s-attrib-rel(entity(1,_417538),entity(20,_417583),_417585^_417588^on(_417585,_417588))
      s-refer(entity(20,_417583))
      s-attrib(entity(20,_417583),_417913^category(_417913,television))

Adding config (from inactive edge):
Active:   modifiers(_310937,_310938)
Inactive: modifier(entity(1,_417538),_417534,_417535)

Adding active edge:
Head: modifiers(_430857,_430858)
Need: modifier(_430857,_430858,_430883)

***duplicate active edge (removed)***

Applying config (0 left, step 11)
---------------------------------
Active:   modifiers(_310937,_310938)
Inactive: modifier(entity(1,_417538),_417534,_417535)

Adding active edge:
Head: modifiers(entity(1,_447114),_446962)
Need: modifiers(entity(1,_447114),_446987)
Acts: s-attrib-rel(entity(1,_447114),entity(20,_447159),_447161^_447164^on(_447161,_447164))
      s-refer(entity(20,_447159))
      s-attrib(entity(20,_447159),_447341^category(_447341,television))

***Topdown addition of inactive entries (1)***
```

```
Adding inactive edge:
Head: modifiers(entity(1,_77618),_77615)

Adding config (from inactive edge):
Active:   modifiers(entity(1,_76524),_76460)
Inactive: modifiers(entity(1,_77618),_77615)

Applying config (0 left, step 12)
---------------------------------
Active:   modifiers(entity(1,_76524),_76460)
Inactive: modifiers(entity(1,_77618),_77615)

Adding inactive edge:
Head: modifiers(entity(1,_91685),_91682)
Acts: s-attrib-rel(entity(1,_91685),entity(20,_91747),_91749^_91752^on(_91749,_91752))
      s-refer(entity(20,_91747))
      s-attrib(entity(20,_91747),_91929^category(_91929,television))

*** Agenda Empty ***

Returning from plan recognizer

<ma>  substitute(p57,p83,V1,_67339,V2)
      V1=modifier(entity(1,_75717),_75713,_75714)
         +-s-attrib-rel(entity(1,_75717),entity(20,_75762),_75764^_75767^on(_75764,_75767))
         +-ref(entity(20,_75762),_75809)
         +-subset(_75713,_75820^ub(_75737,_75741,apply(apply(_75764^_75767^on(_75764,_75767),_75820),
         |  - _75809)),_75714)
         +-refer(entity(20,_75762))
           +-s-refer(entity(20,_75762))
           +-describe(entity(20,_75762))
             +-headnoun(entity(20,_75762),_75930)
             | +-s-attrib(entity(20,_75762),_75964^category(_75964,television))
             | +-subset(_75945,_75964^ub(_75937,_75941,category(_75964,television)),_75930)
             +-modifiers(entity(20,_75762),_75930)
               +-null
      V2=s-attrib-rel(entity(1,_67299),entity(20,_67302),_67304^_67307^on(_67304,_67307))
         s-refer(entity(20,_67302))
         s-attrib(entity(20,_67302),_67328^category(_67328,television))
<ma>  evaluate(V1)
      V1=_205819:refer(entity(1,_67299))
         +-_206195:s-refer(entity(1,_67299))
         +-_206431:describe(entity(1,_67299))
           +-_206625:headnoun(entity(1,_67299),_170995)
           | +-_207092:s-attrib(entity(1,_67299),_168867^category(_168867,creature))
           | +-_207692:subset(_171007,_168867^ub(_170999,_171003,category(_168867,creature)),_170995)
           +-_207825:modifiers(entity(1,_67299),_170995)
             +-_208019:modifier(entity(1,_67299),_170995,_75713)
             | +-_208395:s-attrib(entity(1,_67299),_169008^assessment(_169008,weird))
             | +-_208995:subset(_170995,_172892^ub(_172875,_172879,apply(_169008^assessment(_169008,
             |    - weird),_172892)),_75713)
             +-_209128:modifiers(entity(1,_67299),_75713)
               +-_209322:modifier(entity(1,_67299),_75713,_75714)
               | +-_209698:s-attrib-rel(entity(1,_67299),entity(20,_67302),_67304^_67307^on(_67304,
               | |  - _67307))
               | +-_210389:ref(entity(20,_67302),_75809)
               | +-_210480:subset(_75713,_75820^ub(_75737,_75741,apply(apply(_67304^_67307^on(_67304,
               | |  - _67307),_75820),_75809)),_75714)
               | +-_210571:refer(entity(20,_67302))
               |   +-_210947:s-refer(entity(20,_67302))
               |   +-_211183:describe(entity(20,_67302))
               |     +-_211377:headnoun(entity(20,_67302),_75930)
               |     | +-_211844:s-attrib(entity(20,_67302),_67328^category(_67328,television))
               |     | +-_212444:subset(_75945,_67328^ub(_75937,_75941,category(_67328,television)),
               |     |    - _75930)
               |     +-_212577:modifiers(entity(20,_67302),_75930)
               |       +-_212953:null
               +-_213303:modifiers(entity(1,_67299),_75714)
```

```
                +-_213679:null

Inferred Plan
-------------

_67531:replace-plan(p57)
+-_67597:member(antenna1,[antenna1,fern1])
+-_67602:ref(entity(1,antenna1),antenna1)
+-_67607:construct(modifier(entity(1,antenna1),[antenna1,fern1],[antenna1]),V1,V2)
| V1=modifier(entity(1,_67299),_75713,_75714)
|    +-s-attrib-rel(entity(1,_67299),entity(20,_67302),_67304^_67307^on(_67304,_67307))
|    +-ref(entity(20,_67302),_75809)
|    +-subset(_75713,_75820^ub(_75737,_75741,apply(apply(_67304^_67307^on(_67304,_67307),_75820),
|    |  - _75809)),_75714)
|    +-refer(entity(20,_67302))
|      +-s-refer(entity(20,_67302))
|      +-describe(entity(20,_67302))
|        +-headnoun(entity(20,_67302),_75930)
|        | +-s-attrib(entity(20,_67302),_67328^category(_67328,television))
|        | +-subset(_75945,_67328^ub(_75937,_75941,category(_67328,television)),_75930)
|        +-modifiers(entity(20,_67302),_75930)
|          +-null
| V2=s-attrib-rel(entity(1,_67299),entity(20,_67302),_67304^_67307^on(_67304,_67307))
|    s-refer(entity(20,_67302))
|    s-attrib(entity(20,_67302),_67328^category(_67328,television))
+-_67612:substitute(p57,p83,V1,V2,V3)
 | V1=modifier(entity(1,_67299),_75713,_75714)
 |    +-s-attrib-rel(entity(1,_67299),entity(20,_67302),_67304^_67307^on(_67304,_67307))
 |    +-ref(entity(20,_67302),_75809)
 |    +-subset(_75713,_75820^ub(_75737,_75741,apply(apply(_67304^_67307^on(_67304,_67307),_75820),
 |    |  - _75809)),_75714)
 |    +-refer(entity(20,_67302))
 |      +-s-refer(entity(20,_67302))
 |      +-describe(entity(20,_67302))
 |        +-headnoun(entity(20,_67302),_75930)
 |        | +-s-attrib(entity(20,_67302),_67328^category(_67328,television))
 |        | +-subset(_75945,_67328^ub(_75937,_75941,category(_67328,television)),_75930)
 |        +-modifiers(entity(20,_67302),_75930)
 |          +-null
 | V2=_205819:refer(entity(1,_67299))
 |    +-_206195:s-refer(entity(1,_67299))
 |    +-_206431:describe(entity(1,_67299))
 |      +-_206625:headnoun(entity(1,_67299),_170995)
 |      | +-_207092:s-attrib(entity(1,_67299),_168867^category(_168867,creature))
 |      | +-_207692:subset(_171007,_168867^ub(_170999,_171003,category(_168867,creature)),_170995)
 |      +-_207825:modifiers(entity(1,_67299),_170995)
 |        +-_208019:modifier(entity(1,_67299),_170995,_75713)
 |        | +-_208395:s-attrib(entity(1,_67299),_169008^assessment(_169008,weird))
 |        | +-_208995:subset(_170995,_172892^ub(_172875,_172879,apply(_169008^assessment(_169008,weird)
 |        |    - ,_172892)),_62437)
 |        +-_63139:modifiers(entity(1,_62155),_62437)
 |          +-_63159:modifier(entity(1,_62155),_62437,_62438)
 |          | +-_63189:s-attrib-rel(entity(1,_62155),entity(20,_62158),_62160^_62163^on(_62160,_62163))
 |          | +-_63234:ref(entity(20,_62158),_62531)
 |          | +-_63239:subset(_62437,_62542^ub(_62459,_62463,apply(apply(_62160^_62163^on(_62160,
 |          | |  - _62163),_62542),_62531)),_62438)
 |          | +-_63244:refer(entity(20,_62158))
 |          |   +-_63274:s-refer(entity(20,_62158))
 |          |   +-_63294:describe(entity(20,_62158))
 |          |     +-_63314:headnoun(entity(20,_62158),_62633)
 |          |     | +-_63349:s-attrib(entity(20,_62158),_62184^category(_62184,television))
 |          |     | +-_63389:subset(_62648,_62184^ub(_62640,_62644,category(_62184,television)),_62633)
 |          |     +-_63394:modifiers(entity(20,_62158),_62633)
 |          |       +-_63424:null
 |          +-_63434:modifiers(entity(1,_62155),_62438)
 |            +-_63464:null
 | V3=s-attrib-rel(entity(1,_62155),entity(20,_62158),_62160^_62163^on(_62160,_62163))
```

```
|    s-refer(entity(20,_62158))
|    s-attrib(entity(20,_62158),_62184^category(_62184,television))
+-_62378:evaluate(V1)
| V1=_62894:refer(entity(1,_62155))
|    +-_62924:s-refer(entity(1,_62155))
|    +-_62944:describe(entity(1,_62155))
|      +-_62964:headnoun(entity(1,_62155),_62784)
|      | +-_62999:s-attrib(entity(1,_62155),_62761^category(_62761,creature))
|      | +-_63039:subset(_62790,_62761^ub(_62786,_62788,category(_62761,creature)),_62784)
|      +-_63044:modifiers(entity(1,_62155),_62784)
|        +-_63064:modifier(entity(1,_62155),_62784,_62437)
|        | +-_63094:s-attrib(entity(1,_62155),_62762^assessment(_62762,weird))
|        | +-_63134:subset(_62784,_62847^ub(_62838,_62840,apply(_62762^assessment(_62762,weird),
|        |    - _62847)),_62437)
|        +-_63139:modifiers(entity(1,_62155),_62437)
|          +-_63159:modifier(entity(1,_62155),_62437,_62438)
|          | +-_63189:s-attrib-rel(entity(1,_62155),entity(20,_62158),_62160^_62163^on(_62160,_62163))
|          | +-_63234:ref(entity(20,_62158),_62531)
|          | +-_63239:subset(_62437,_62542^ub(_62459,_62463,apply(apply(_62160^_62163^on(_62160,
|          | |  - _62163),_62542),_62531)),_62438)
|          | +-_63244:refer(entity(20,_62158))
|          |   +-_63274:s-refer(entity(20,_62158))
|          |   +-_63294:describe(entity(20,_62158))
|          |     +-_63314:headnoun(entity(20,_62158),_62633)
|          |     | +-_63349:s-attrib(entity(20,_62158),_62184^category(_62184,television))
|          |     | +-_63389:subset(_62648,_62184^ub(_62640,_62644,category(_62184,television)),_62633)
|          |     +-_63394:modifiers(entity(20,_62158),_62633)
|          |       +-_63424:null
|          +-_63434:modifiers(entity(1,_62155),_62438)
|            +-_63464:null
+-_62383:s-reject(p57,V1)
  V1=s-attrib-rel(entity(1,_62155),entity(20,_62158),_62160^_62163^on(_62160,_62163))
     s-refer(entity(20,_62158))
     s-attrib(entity(20,_62158),_62184^category(_62184,television))

User Model: Adding Proposition
plan(user,p129,mb(system,user,replace(p57,V1)))
V1=_62894:refer(entity(1,_62155))
   +-_62924:s-refer(entity(1,_62155))
   +-_62944:describe(entity(1,_62155))
     +-_62964:headnoun(entity(1,_62155),_62784)
     | +-_62999:s-attrib(entity(1,_62155),_62761^category(_62761,creature))
     | +-_63039:subset(_62790,_62761^ub(_62786,_62788,category(_62761,creature)),_62784)
     +-_63044:modifiers(entity(1,_62155),_62784)
       +-_63064:modifier(entity(1,_62155),_62784,_62437)
       | +-_63094:s-attrib(entity(1,_62155),_62762^assessment(_62762,weird))
       | +-_63134:subset(_62784,_62847^ub(_62838,_62840,apply(_62762^assessment(_62762,weird),_62847))
       |    - ,_62437)
       +-_63139:modifiers(entity(1,_62155),_62437)
         +-_63159:modifier(entity(1,_62155),_62437,_62438)
         | +-_63189:s-attrib-rel(entity(1,_62155),entity(20,_62158),_62160^_62163^on(_62160,_62163))
         | +-_63234:ref(entity(20,_62158),_62531)
         | +-_63239:subset(_62437,_62542^ub(_62459,_62463,apply(apply(_62160^_62163^on(_62160,_62163),
         | |  - _62542),_62531)),_62438)
         | +-_63244:refer(entity(20,_62158))
         |   +-_63274:s-refer(entity(20,_62158))
         |   +-_63294:describe(entity(20,_62158))
         |     +-_63314:headnoun(entity(20,_62158),_62633)
         |     | +-_63349:s-attrib(entity(20,_62158),_62184^category(_62184,television))
         |     | +-_63389:subset(_62648,_62184^ub(_62640,_62644,category(_62184,television)),_62633)
         |     +-_63394:modifiers(entity(20,_62158),_62633)
         |       +-_63424:null
         +-_63434:modifiers(entity(1,_62155),_62438)
           +-_63464:null

User Model: Adding Proposition
contributed(user,p129)
```

```
User Model: Adding Proposition
evaluation(p129,success)

==========
Checking Rules
==========
Adding beliefs about the inferred plan

Applying rule4
User Model: Adding Proposition
replace(p57,V1)
V1=_361371:refer(entity(1,_361386))
   +-_361404:s-refer(entity(1,_361386))
   +-_361423:describe(entity(1,_361386))
     +-_361442:headnoun(entity(1,_361386),_361455)
     | +-_361483:s-attrib(entity(1,_361386),_361501^category(_361501,creature))
     | +-_361546:subset(_361479,_361501^ub(_361465,_361472,category(_361501,creature)),_361455)
     +-_361565:modifiers(entity(1,_361386),_361455)
       +-_361585:modifier(entity(1,_361386),_361455,_361599)
       | +-_361620:s-attrib(entity(1,_361386),_361638^assessment(_361638,weird))
       | +-_361683:subset(_361455,_361690^ub(_361609,_361616,apply(_361638^assessment(_361638,weird),
       |    - _361690)),_361599)
       +-_361708:modifiers(entity(1,_361386),_361599)
         +-_361728:modifier(entity(1,_361386),_361599,_361742)
         | +-_361763:s-attrib-rel(entity(1,_361386),entity(20,_361783),_361785^_361788^on(_361785,
         | |  - _361788))
         | +-_361844:ref(entity(20,_361783),_361848)
         | +-_361855:subset(_361599,_361862^ub(_361752,_361759,apply(apply(_361785^_361788^on(_361785,
         | |  - _361788),_361862),_361848)),_361742)
         | +-_361886:refer(entity(20,_361783))
         |   +-_361919:s-refer(entity(20,_361783))
         |   +-_361938:describe(entity(20,_361783))
         |     +-_361957:headnoun(entity(20,_361783),_361970)
         |     | +-_361998:s-attrib(entity(20,_361783),_362016^category(_362016,television))
         |     | +-_362061:subset(_361994,_362016^ub(_361980,_361987,category(_362016,television)),
         |     |    - _361970)
         |     +-_362080:modifiers(entity(20,_361783),_361970)
         |       +-_362121:null
         +-_362143:modifiers(entity(1,_361386),_361742)
           +-_362184:null

--------
Evaluate
--------

postponing evaluation of mb(system,user,category(_64327,creature))
<ma>  subset([fern1,antenna1,corner1,television1],_64442^ub(system,user,category(_64442,creature)),
      - _64396)
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
<ma>  subset([antenna1,fern1],_64631^ub(system,user,assessment(_64631,weird)),_64540)
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
postponing evaluation of mb(system,user,on(_64327,_64724))
<ma>  ref(entity(20,_64724),_64789)
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
postponing evaluation of mb(system,user,on(_64327,_64724))
postponing evaluation of subset([antenna1,fern1],_64803^ub(system,user,on(_64803,_64724)),_64683)
postponing evaluation of mb(system,user,category(_64724,television))
<ma>  subset([fern1,antenna1,corner1,television1],_64957^ub(system,user,category(_64957,television)),
      - _64911)
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
postponing evaluation of mb(system,user,on(_64327,_64724))
postponing evaluation of subset([antenna1,fern1],_64803^ub(system,user,on(_64803,_64724)),_64683)
postponing evaluation of mb(system,user,category(_64724,television))
<con> [television1]=[_64724]
```

```
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
postponing evaluation of mb(system,user,on(_64327,television1))
<ma>  subset([antenna1,fern1],_64803^ub(system,user,on(_64803,television1)),_64683)
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
postponing evaluation of mb(system,user,on(_64327,television1))
<con> mb(system,user,category(television1,television))
postponing evaluation of mb(system,user,category(_64327,creature))
postponing evaluation of mb(system,user,assessment(_64327,weird))
postponing evaluation of mb(system,user,on(_64327,television1))
<con> [antenna1]=[_64327]
<con> mb(system,user,category(antenna1,creature))
<con> mb(system,user,assessment(antenna1,weird))
<con> mb(system,user,on(antenna1,television1))


removing plan p57 from belief space

User Model: Adding Proposition
contributed(user,p145)

User Model: Adding Proposition
evaluation(p145,success)

User Model: Adding Proposition
mgoal(system,user,p145,knowref(system,entity(1,_182682)))

*** mgoal updated ***

Applying rule5
User Model: Adding Proposition
achieve(p145,knowref(system,entity(1,antenna1)))
```

A.5 Constructing "Okay"

```
==============
Checking Rules
==============
Adding goals

Applying rule9
User Model: Adding Proposition
goal(system,mb(system,user,achieve(p145,knowref(system,entity(1,antenna1)))))

==========
Responding
==========

Goal   : mb(system,user,achieve(p145,knowref(system,entity(1,antenna1))))
Effect: bel(user,goal(system,mb(system,user,achieve(p145,knowref(system,entity(1,antenna1))))))

---------
Construct
---------

Initial Setup
-------------
Trying: accept-plan(p145)
<con> achieve(p145,knowref(system,entity(1,antenna1)))
<con> plan(user,p145,knowref(system,entity(1,antenna1)))
New edge added

End of Initialization
---------------------
```

```
Expanding (step 0  active 1)
----------------------------
Head: accept-plan(p145)
Need: s-accept(p145)
Trying: s-accept(p145)
New edge added


Constructed Plan
----------------
accept-plan(p145)
+-s-accept(p145)

Actions to be fed to generator
------------------------------
s-accept(p145)

User Model: Adding Proposition
plan(system,p207,mb(system,user,achieve(p145,knowref(system,entity(1,antenna1)))))

User Model: Adding Proposition
contributed(system,p207)

User Model: Adding Proposition
evaluation(p207,success)

===============
Checking Rules
===============
Adding beliefs about the constructed plan

===============
Checking Rules
===============
Adding goals
```