Resolving Plan Ambiguity for Cooperative Response Generation

Peter van Beek
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
vanbeek@cs.ualberta.ca

Robin Cohen
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
rcohen@dragon.waterloo.edu

Abstract

Recognizing the plan underlying a query aids in the generation of an appropriate response. In this paper, we address the problem of how to generate cooperative responses when the user's plan is ambiguous. We show that it is not always necessary to resolve the ambiguity, and provide a procedure that estimates whether the ambiguity matters to the task of formulating a response. If the ambiguity does matter, we propose to resolve the ambiguity by entering into a clarification dialogue with the user and provide a procedure that performs this task. Together, these procedures allow a question-answering system to take advantage of the interactive and collaborative nature of dialogue in recognizing plans and resolving ambiguity.

1 Introduction

Somewhat obviously, plan recognition is the process of inferring an agent's plan from observation of the agent's actions. The agent's actions can be physical actions or speech actions. Four principal methods for plan recognition have been proposed in the literature. The methods are plausible inference [Allen, 1983; Carberry, 1988; Litman and Allen, 1987; Sidner, 1985], parsing [Vilain, 1990], circumscribing a hierarchical representation of plans and using deduction [Kautz, 1987], and abduction [Charniak and McDermott, 1985; Konolige and Pollack, 1989; Poole, 1989].

Our particular interest is in the use of plan recognition in question-answering systems, where recognizing the plan underlying a user's queries aids in the generation of an appropriate response. Here, the plan of the user, once recognized, has been used to: supply more information than is explicitly requested [Allen, 1983; Luria, 1987], handle pragmatically ill-formed queries [Carberry, 1988], provide an explanation from the appropriate perspective [McKeown *et al.*, 1985], respond to queries that result from an invalid plan [Pollack, 1984; Pollack, 1986], and avoid misleading responses and produce user-specific cooperative responses [Joshi *et al.*, 1984; van Beek, 1987; Cohen *et al.*, 1989].

Example 1 ([Joshi *et al.*, 1984]). As an example of a cooperative response consider the following exchange between student and student-advisor system. The plan of the student is to avoid failing a course by dropping it.

User: Can I drop numerical analysis?

System: Yes, but you will still fail the course since your mark will be recorded as withdrawal while failing.

If the system just gives the direct answer, Yes, the student will remain unaware that the plan is faulty. The more cooperative answer warns the student.

An important weakness of this work in response generation, however, is the reliance on a plan recognition component being able to uniquely determine the plan of the user. This is clearly too strong an assumption as the user's actions often will be consistent with more than one plan, especially after only one or a few utterances when there is insufficient context to help decide the plan of the user. In Example 1 there are many reasons why a student may want to drop a course, such as resolving a scheduling conflict, avoiding failing the course, or finding the material uninteresting. There may be no reason to prefer one alternative over the other, yet we may still want to generate a response that does more than just give a direct answer to the user's query.

In this paper, we address the problem of what the system should do when the user's actions are ambiguous as they are consistent with more than one plan. Much previous work has considered heuristics that allow the plan recognition system, given an assessment of the context and dialogue so far, to prefer some plans over others (e.g. [Allen, 1983; Carberry, 1988; McKeown *et al.*, 1985]). However, we argue that, unless we are willing to sometimes arbitrarily commit to one plan instead of another, there will be times when one plan cannot be chosen over another and therefore there will be ambiguity about which plan the user is pursuing. As a result, we also need other methods to resolve the ambiguity. Existing proposals for resolving ambiguity beyond heuristics are underspecified and what usually underlies these proposals is the assumption that we always want to determine one unique plan [Carberry, 1988; Litman and Allen, 1987; Sidner, 1985][1].

[1]For example, Litman and Allen [1987, p. 10] give two ways for discriminating between the possible plans when heuristics cannot eliminate all but one plan, "... if it is the

We show how to relax the assumption that the plan recognition component returns a single plan. That is, given that the result of the plan recognition phase will usually be a disjunction of possible plans, we show how to design a response component to generate cooperative responses given the disjunction. We show that it is not always necessary to resolve ambiguity, and provide a procedure that allows the response component to estimate whether the ambiguity matters to the task of formulating a response. If the ambiguity does not matter, the response component can continue to answer the user's queries and ignore the ambiguity in the underlying plan until further queries help clarify which plan the user is pursuing. If the ambiguity does matter, the system should take advantage of the interactive and collaborative nature of dialogue in recognizing plans and resolving ambiguity. A key contribution of this work therefore is providing a clear criterion for *when* to respond to a question with a question that will differentiate between some of the possibilities. We also propose a specific solution to *what* questions should then be asked of the user. Moreover, these questions are asked only to resolve the ambiguity to the point where it no longer matters (this is not necessarily to a unique plan).

Our solution makes use of the critiquing of possible plans and identifies plans with the same fault. Questions are asked to prune sets of these plans. After sufficient pruning, all remaining plans are annotated with the same fault, and thus the ambiguity does not matter. We argue that this approach is preferable to proposing one plan and resorting to debugging when an incorrect plan is chosen.

Example 2. Here are two examples to give a flavor of what we are proposing. There are two agents: a cook and an expert who is cooperative, helpful, and adept at recognizing plans.

a. Suppose the cook says to the expert, "I'm making marinara sauce. Is a red wine a good choice?" The expert recognizes the cook could be pursuing three possible plans: make fettucini marinara or spaghetti marinara (both a pasta dish) or chicken marinara (a meat dish). The expert has the criteria for wine selection that red wine should be served if the meal is chicken, fettucini marinara, or spaghetti marinara and white if fettucini alfredo. There is enough information for the expert to decide red wine should be bought and the ambiguity does not need to be resolved to cooperatively answer the question.

b. Now suppose the expert also knows that the cook's dinner guest is a vegetarian and so would not be able to eat if a meat dish was served. Here the ambiguity is important as the expert has recognized that the cook's plan to entertain his guest may be faulty. The expert will want to resolve the ambiguity enough to be assured that the proposed meal does not include a meat dish and so clarifies this with the cook.

[system's] turn in the dialogue ..., then the [system] may initiate a clarification subdialogue. If it is still the [user's] turn, the [system] may wait for further dialogue to distinguish between the possibilities." However, this proposal is never developed further.

2 Estimating Whether the Ambiguity Matters

Example 2, above, showed that sometimes it is necessary to resolve ambiguity and sometimes it is not. Here we give criteria for judging which is the case. The result is a procedure that allows the response component to estimate whether the ambiguity matters to the task of formulating a response.

Assuming we can answer the user's query, deciding when we want to give more than just a direct answer to the user's query depends on the plan of the user. For example, a cooperative response should warn a user that a plan will fail because some of its preconditions are not satisfied [Allen, 1983] or if there exists another plan that has the same effects but is in some sense a better plan [Pollack, 1984; Joshi *et al.*, 1984]. Our algorithm for generating a response even when the plan of the user is ambiguous involves first a plan recognition phase and then a call to a procedure to critique the set of possible plans from the plan recognition phase (see Fig. 2). As a result, the plans are annotated with their faults. The catalogue of possible plan annotations used in the examples in this paper is shown below.

1. Failure of preconditions: *preconditions.*
2. Temporally inconsistent.
3. There exists a better plan: *primitive actions.*
4. Faultless.

We have restricted the catalogue of annotations to those faults that are easy to detect (see the discussion below on the representation of the libraries of typical plans of action used in the plan recognition phase). The catalogue could, of course, be expanded as much previous work has identified different kinds of faults in a plan that a cooperative response should warn a user about (e.g. [Allen, 1983; Pollack, 1986; Joshi *et al.*, 1984; Quilici *et al.*, 1988; van Beek, 1987; Luria, 1987] and see [Calistri, 1990]). For example, Joshi et al. [Joshi *et al.*, 1984] discuss the case where a user's plan fails but there exists an alternative plan that does not fail and in [van Beek, 1987] we show that the user may have *competing* goals that need to be addressed in a cooperative response. However, the search for faults in plans must be balanced against the need for a timely response and how many additional faults can be searched for is currently left for future work.

Once the plans have been critiqued, we can estimate whether the ambiguity matters. The deciding criterion for estimating whether the ambiguity matters is whether the plans have the same annotation (see the catalogue of possible annotations shown above; the procedure is shown in Fig. 2). Briefly, there are two cases: (1) the plans are all faultless (Case 1a of procedure *Ambiguity_Matters*) and (2) the plans are all annotated with the same fault (Case 1b of procedure *Ambiguity_Matters*).

To make the examples concrete, we adopt Kautz's [1987] theory of plan recognition and representation for libraries of plans. Briefly, a library of typical plans of action is represented as a set of first-order predicate calculus statements called an event hierarchy. Fig. 1 shows

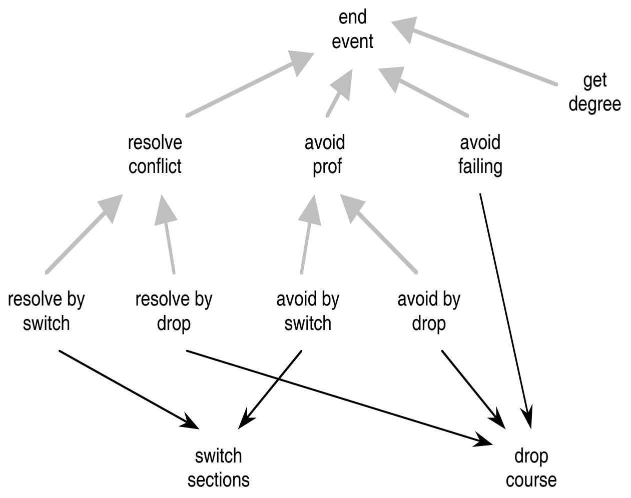


Figure 1: Plan library for course-advising examples

a graphical representation of a part of the plan library for the course-advising domain. The thick, grey arrows represent abstraction (or "isa") links and the thin, black arrows represent decomposition links. Preconditions, equality constraints, and temporal constraints are not shown in the graphical representation but are also part of the knowledge we must represent. As well, we must represent under what conditions one plan is better than another plan. It is these constraints that are checked in the plan critiquing phase.

As an example of determining whether the ambiguity matters to the task of formulating a cooperative response, consider the exchanges between student and a student-advisor system shown in (3) below. Given the student's query (3a), procedure *Response_Generation* in Fig. 2 shows the steps in formulating a response. As a first step, the plan recognition procedure is called to determine the set of possible plans of the user. Given the plan library in Fig. 1, the underlying plan of the student is ambiguous. The two possible plans are as shown in (3). Procedure *Critique* is called to critique the plans. One precondition of the action *switch-sections(FromSection, ToSection)* is that there is space available in the destination section.

3a. U: Can I switch to the other section of my numerical analysis course?
Possible plans:
1. Avoid uninteresting prof by switching sections of course.
2. Resolve scheduling conflict by switching sections of course.
b. S: Yes.
c. S: No, there is no space available.

For the response shown in (3b), suppose that there is room for the student and thus the precondition is satisfied. In this example, the result of plan critiquing is that both plans are labeled with the annotation "Faultless." Procedure *Ambiguity_Matters* is called to determine whether the ambiguity regarding the plan of the user matters to the task of formulating a response. It is found that the ambiguity does not matter as both plans are faultless (Case 1a of the algorithm). Finally, the critiqued plans are used in formulating a response to the original query.

For the response shown in (3c), suppose that there is no room for the student and thus the precondition is not satisfied. The result of plan critiquing is that both plans are labeled with the annotation "Failure of preconditions: *space-available(ToSection)*." Procedure *Ambiguity_Matters* is called to determine whether the ambiguity regarding the plan of the user matters to the task of formulating a response. It is found that the ambiguity does not matter as both plans are annotated with the same fault (Case 1b of the algorithm). Finally, the critiqued plans are used in formulating a response to the original query.

In general, in (Case 1a) a response generation procedure can just give a direct answer to the user's query, and in (Case 1b) can give a direct answer plus any warranted additional information, such as telling the user about the fault.

In the above examples it was found that the ambiguity did not matter as there was enough information to generate a cooperative response. If instead it were found that the ambiguity did matter (Case 2 of the algorithm) we propose that we enter into a clarification dialogue with the user to resolve the ambiguity to the point where it no longer does matter, i.e., until we are in (Case 1). The remaining plans would then be used in formulating a response.

3 Clarification Dialogues

What should we ask the user when a clarification is necessary? Clearly, we do not want to simply list the set of possible plans and ask which is being pursued. The procedure to determine what to say (*Clarify*) is shown in Fig. 2. Our proposal for clarification dialogues is tied to a hierarchical plan library. The input to the algorithm is a set of possible plans that relate the user's action to the top-level or end event. Each plan in the set is annotated with a critique. The key idea is to ask about the highest level possible, check whether the ambiguity still needs to be further resolved, and if so, ask at the next level down, iteratively, through the hierarchy of events. The algorithm groups the plans according to the type of fault the plan is annotated with and chooses the group with the fewest events in it and asks first about that.

As an example of where the ambiguity matters and we ask a question to clarify, consider the exchanges between student and a student-advisor system shown in (4). Given the plan library in Fig. 1, the underlying plan of the student is ambiguous. The three possible plans and the results of plan critiquing are as shown.

It is found that the ambiguity matters as the annotations are different for some of the plans (Case 2b of procedure *Ambiguity_Matters*). As a result, procedure *Clarify* is called to initiate a clarification dialogue. At the start of the procedure, *current_level* is initialized to

```
procedure Response_Generation(Query)
begin
    $S \leftarrow$ Plan_Recognition(Query)
    $S \leftarrow$ Critique(S)
    if Ambiguity_Matters(S) = "Yes" then
        $S \leftarrow$ Clarify(S)
    Generate response based on query and plan(s) in $S$
end

procedure Critique(S)
begin
    for each plan in $S$ do
        critique and annotate plan
    return($S$)
end

procedure Ambiguity_Matters(S)
begin
    We are in one of the following two cases:
    Case 1. The ambiguity does not matter.
        The critiques are the same for all the plans. I.e.,
        a. every plan is faultless, or
        b. every plan is annotated with the same fault.
        return("No")
    Case 2. The ambiguity does matter.
        The critiques are different for some or all of the
        plans. I.e.,
        a. some, but not all, of the plans are faultless, or
        b. every plan is annotated with a fault and the
          faults are not all the same.
        return("Yes")
end

procedure Clarify(S)
begin
    current_level $\leftarrow$ first disjunctive branch point from
        the top in $S$.
    Partition $S$ into the set of sets $\{F_1, \ldots, F_k\}$,
        assigning two plans to the same $F_i$ if and only
        if they have the same fault annotation.
    while Ambiguity_Matters = "Yes" do
    begin
        $F_{min} \leftarrow F_i$ in $S$ with fewest distinct events
            one level below current_level
        List distinct events in $F_{min}$ one level below
            current_level and ask user whether one of
            the events is being pursued
        if user's answer = "Yes" then $S \leftarrow F_{min}$
        else $S \leftarrow S - F_{min}$
        current_level $\leftarrow$ next branch point in $S$
    end
    return($S$)
end
```

Figure 2: Response generation algorithm

be end-event. (We have only given English descriptions of the possible plans in this example, but see Fig. 4 for the and/or graph that represents the possible plans that arise in a cooking example.) The set of possible plans is partitioned into $\{F_1, F_2\}$ where F_1 contains plan (4a.1) and F_2 contains plan (4a.2) and plan (4a.3). Next, F_{min} is found to be F_1 as it has the smallest number of distinct events one level below *current_level*. Asking about the event one level below *current_level* leads to the question shown in (4b). Depending on whether the answer is no (4c) or yes (4e), the set of possible plans, S, is updated appropriately. In both cases it is found that the ambiguity no longer matters. Finally, the remaining set of critiqued plans are used in formulating a response to the original query (4d or 4f). In this example, by carefully choosing the question to ask, we are able to resolve the ambiguity by asking the user a single question about the single event "avoid-failing."

4a. U: Can I drop numerical analysis?
Possible plans with annotations:
1. Avoid failing by dropping course.
Failure of preconditions: *not failing(Student).*
2. Avoid uninteresting prof by dropping course.
There exists a better plan:
switch-sections(FromSection, ToSection).
3. Resolve scheduling conflict by dropping course.
There exists a better plan:
switch-sections(FromSection, ToSection).
b. S: Are you trying to avoid failing the course?
c. U: No.
d. S: You can drop the course, but if you are trying to avoid an uninteresting professor or trying to resolve a scheduling conflict, a better way is to switch to another section.
e. U: Yes.
f. S: You can drop the course, but you will still fail the course since your mark will be recorded as withdrawal while failing.

As a second example, recall our story in the introduction about cooking for our vegetarian guest. Given the plan library in Fig. 3, the underlying plan of the cook is ambiguous. The three possible plans are as shown in (5) and, in more detail, in Fig. 4.

5a. U: I'm making marinara sauce. Is a red wine a good choice?
Partial description of possible plans with annotations (see Fig. 4):
1. Make pasta dish by making fettucini marinara.
Faultless.
2. Make pasta dish by making spaghetti marinara.
Faultless.
3. Make meat dish by making chicken marinara.
Failure of preconditions: *not vegetarian(Guest).*
b. S: Are you making a meat dish?

Using procedure *Clarify*, the three plans are critiqued and it is found that the ambiguity matters. The two plans involving a pasta dish are found to be faultless but the plan involving a meat dish is found to be faulty as a precondition is false. Using procedure *Clarify*, the question asked to resolve the ambiguity would be "Are you making a meat dish (perhaps with justification of why we are asking)?" After either answer of yes or no

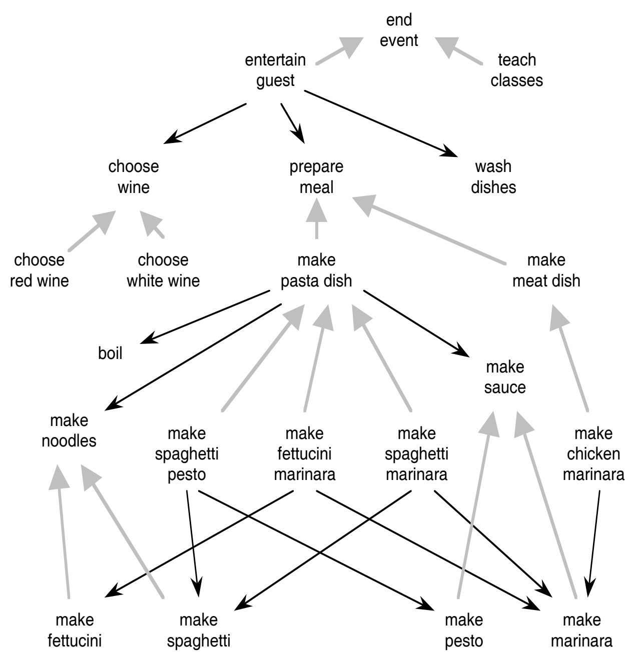


Figure 3: Plan library for cooking examples ([Kautz, 1987]; modified)

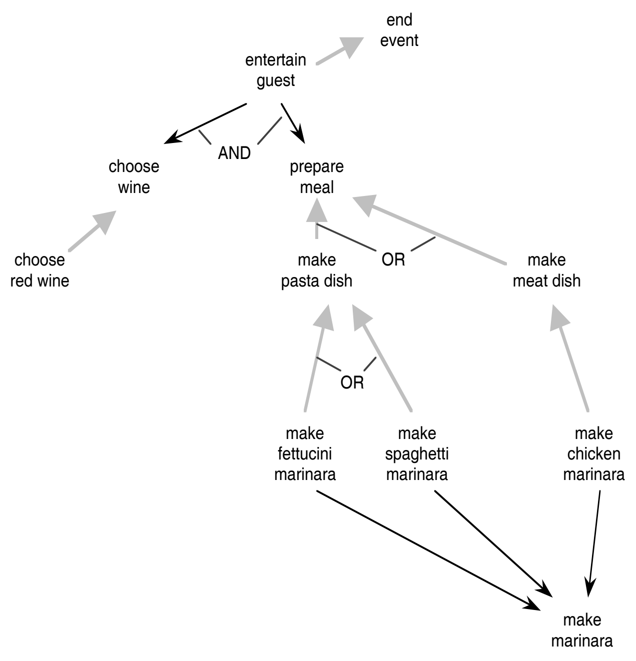


Figure 4: Possible plans for cooking example

we know enough that the ambiguity no longer matters. Note that if we just ask the more general "What are you making?" this allows such uninformative responses as "dinner" or just "you'll see."

As the examples show, when asking a question we propose to ask about as high a level of event as possible that still helps to resolve the ambiguity, and to work top down. Starting with the top most events and working down may sometimes give as many or more questions as bottom up approaches. However, others have noted that bottom up dialogues are more complex to understand [Cohen, 1983, p. 25] and more liable to misinterpretation [Carberry, 1985, p. 54]. Therefore, we believe that a top down approach is preferable[2].

If more than one question is necessary to resolve the ambiguity, we want the questions to not appear to be disjointed. This is achieved in our proposal by asking about as high a level of event as possible that still helps to resolve the ambiguity and moving systematically downward through the hierarchy of possible plans and thus having the "focus" of the questions change gradually.

The design of the algorithm takes into consideration that it is important to attempt to minimize both the length of clarification dialogues and the length of questions. Fortunately, because plans are hierarchical, we can often ask about a group of plans by asking about a single event. For example, with reference to Fig. 4 and procedure *Clarify*, if some F_i contained the plans: make spaghetti pesto, make fettucini marinara, and make spaghetti marinara, the algorithm would ask about this set of plans by asking the user, "Are you making a pasta dish?" In addition, the questions asked all result in yes/no answers, which removes the task of disambiguating the user's response. An open problem for future research is developing tools for comparing criteria for generating questions. For instance, we may want to choose the group of plans to ask about as a combination of how large a question it will give and how probable or likely the plans in the group actually are.

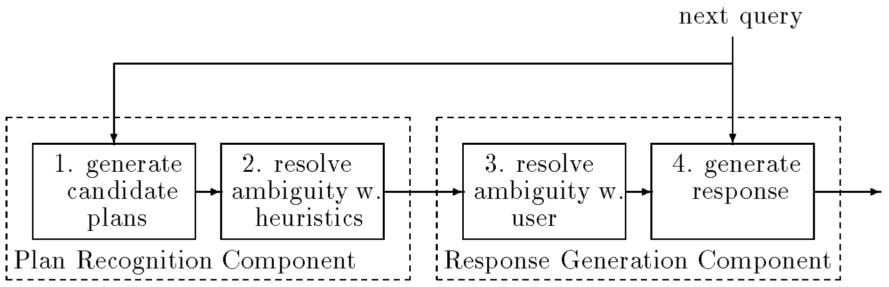
[2]Note that, independent of the order that questions are asked, some questions can be eliminated using plans known from a previous discourse or background knowledge about the user. For example, in a student-advising domain, we would not want to ask a user what degree they are pursuing every time they used the system.

4 Discussion

In this section we summarize our proposals and defend our position that this straightforward way of doing things is a good way. With reference to Fig. 5, we discuss the design of boxes 2, 3, and 4 and the tradeoffs involved between boxes 2 and 3.

Box 2: Resolve the ambiguity with heuristics. As mentioned earlier, many researchers have proposed heuristics to prefer one plan over another [Allen, 1983; Carberry, 1988; McKeown *et al.*, 1985; Goldman and Charniak, 1988; Neufeld, 1989; Kautz, 1987]. Some of these heuristics can be incompatible with cooperative response generation. For example, Allen's [1983] preference heuristics are generally incompatible with recognizing and responding to faulty plans. Because we are using plan recognition for response generation, this should affect the design of Box 2 and therefore what gets passed to Box 3.

Box 3: Resolve the ambiguity with the user. Previous work in response generation makes the assumption that what gets passed to the RG component is a



next query
1. generate candidate plans
2. resolve ambiguity w. heuristics
3. resolve ambiguity w. user
4. generate response
Plan Recognition Component
Response Generation Component

Figure 5: Major modules of query-answering system

single plan the PR component proposes the user is pursuing. We argue that, unless we are willing to sometimes arbitrarily commit to one plan instead of another, there will be times when one plan cannot be chosen over another and therefore there will be ambiguity about which plan the user is pursuing. Result: We need other methods, in addition to heuristics, to resolve the ambiguity. In plan recognition in a discourse setting (as opposed to key-hole recognition), the plan the user is pursuing is knowable simply by asking the user. But we do not want to always just ask if it is not necessary so we need to know when to start a clarification dialogue and what to say. To this end, Box 3 contains a procedure that estimates by plan critiquing whether the ambiguity matters to the task of formulating a response. If the ambiguity does not matter the result is passed to Box 4. If the ambiguity does matter, a procedure is called that starts a clarification dialogue, responding to the user's question with questions that iteratively differentiate between the possibilities.

Box 2 vs. Box 3: The tradeoffs. Much previous work in plan recognition makes the assumption that we want the PR component to commit to and return a single plan. Carberry [1988] and McKeown [1985], for example, use a strong heuristic to commit to a single plan. However, committing to a single plan means the system will at times commit to the wrong plan and thus will require the ability to handle natural language debugging dialogues with the user to correct itself. Carberry [1989, p. 4] argues that a system will appear "unintelligent, obtuse, and uncooperative" if it engages in lengthy clarification dialogues. However, a procedure to perform a debugging dialogue is not specified and is, we speculate, a difficult problem. We argue for not committing early. Our hypothesis is that a clarification dialogue is better than a debugging dialogue. The questions in the clarification dialogues are simple to answer, whereas determining that the system has misunderstood your plan requires users to engage in plan recognition. That is, users must recognize the plan the RG component is using from its responses and note that it differs from their plans. Moreover, the user may not recognize the system is wrong and be misled. Finally, we argue that, if the questions are carefully chosen, the clarification dialogues need not be lengthy or too frequent.

Preference heuristics can still be used in our approach. These would best be applied when too many top level events give an unwieldy clarification question. There may be tradeoffs between overcommitting in the plan recognition process and engaging in lengthy clarification dialogue, particularly with a large set of complex candidate plans. This may suggest applying pruning heuristics more actively in the plan recognition process (Box 2) to reduce the number of questions asked in the clarification dialogue (Box 3). For future work, these tradeoffs will be examined more closely as we test the algorithms more extensively.

Box 4: Generate the response. Once Box 3 has estimated that any remaining ambiguity does not matter to generating a cooperative response, the disjunction of possible plans is passed to Box 4. Generating a response is handled by a procedure described in [van Beek, 1987]. There are two cases.

1. Every plan is faultless, so we just give a direct answer to the user's query and ignore the underlying plan until further queries help clarify which plan the user is pursuing.
2. Every plan has the same fault, so we give a direct answer plus some additional information that warns the user about the deficiency and perhaps suggests some alternatives (see [Joshi *et al.*, 1984; van Beek, 1987]).

5 Future Work and Conclusion

This paper makes contributions to the areas of plan recognition and response generation, as follows. For plan recognition, we have shown that there are cases where it is possible to retain a disjunction of possible plans and avoid the work incurred in the application of certain pruning heuristics which propose a single plan, when the plan recognition is being done for the purpose of generating a cooperative response. This also demonstrates that it is possible to use Kautz-style plan recognition for cooperative response generation. Our solution requires initially determining whether it is important to resolve plan ambiguity. And this involves critiquing the full set of possible plans, rather than the one plan that would be proposed if full pruning heuristics were applied. We argue that the process of critiquing, as described in this paper, is not difficult to implement (see [van Beek, 1987] for a description of the implementation) and incurs little overhead and would thus argue that there is a cost saving overall.

The paper also contributes to response generation by providing clear criteria for the initiation of clarification dialogues. Furthermore, we provide an initial proposal for what questions to ask during clarification, and argue for the advantages of our top-down approach. We believe that natural language generation systems should be designed to involve the user more directly and are clearly demonstrating that this is plausible.

We acknowledge that determining the ideal set of questions to ask is still open to future research. This could be empirically tested by observing human advice givers or by gauging reactions to certain clarification dialogues from human audiences. Our preference is to first defend our design decision on theoretical grounds, arguing that if the questions we supply are not "optimal" in terms of work spent to generate and need for further clarification, they are only marginally less "optimal" than other options which could be followed.

Acknowledgements. We thank Fei Song and Bruce Spencer for comments on an earlier version of this paper and for many discussions about plan recognition. Financial assistance was received from the Central Research Fund of the University of Alberta and the Natural Sciences and Engineering Research Council of Canada.

References

[Allen, 1983] J. F. Allen. Recognizing intentions from natural language utterances. In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*, pages 107–166. MIT Press, 1983.

[Calistri, 1990] R. J. Calistri. *Classifying and Detecting Plan-Based Misconceptions for Robust Plan Recognition.* PhD thesis, Brown Univ., 1990. Available as: Dept. of Computer Science Technical Report CS-90-11.

[Carberry, 1985] S. Carberry. *Pragmatic Modeling in Information System Interfaces.* PhD thesis, Univ. of Delaware, 1985. Available as: Dept. of Computer and Information Sciences Technical Report 86-07.

[Carberry, 1988] S. Carberry. Modeling the user's plans and goals. *Computational Linguistics*, 14:23–27, 1988.

[Carberry, 1989] S. Carberry. A new look at plan recognition in natural language dialogue. Dept. of Computer and Information Sciences Technical Report 90-08, Univ. of Delaware, 1989.

[Charniak and McDermott, 1985] E. Charniak and D. V. McDermott. *Introduction to Artificial Intelligence.* Addison Wesley, 1985.

[Cohen *et al.*, 1989] R. Cohen, M. Jones, A. Sanmugasunderam, B. Spencer, and L. Dent. Providing responses specific to a user's goals and background. *Int. Journal of Expert Systems: Research and Applications*, 2:135–162, 1989.

[Cohen, 1983] R. Cohen. *A Computational Model for the Analysis of Arguments.* PhD thesis, Univ. of Toronto, 1983.

[Goldman and Charniak, 1988] R. Goldman and E. Charniak. A probabilistic assumption-based truth maintenance system for plan recognition. In *Proc. of the AAAI-88 Workshop on Plan Recognition*, St. Paul, Minn., 1988.

[Joshi *et al.*, 1984] A. Joshi, B. Webber, and R. Weischedel. Living up to expectations: Computing expert responses. In *Proc. of the Fourth National Conf. on Artificial Intelligence*, pages 169–175, Austin, Tex., 1984.

[Kautz, 1987] H. A. Kautz. *A Formal Theory of Plan Recognition.* PhD thesis, Univ. of Rochester, 1987. Available as: Dept. of Computer Science Technical Report 215.

[Konolige and Pollack, 1989] K. Konolige and M. E. Pollack. Ascribing plans to agents. In *Proc. of the Eleventh Int. Joint Conf. on Artificial Intelligence*, pages 924–930, Detroit, Mich., 1989.

[Litman and Allen, 1987] D. J. Litman and J. F. Allen. A plan recognition model for subdialogue in conversations. *Cognitive Science*, 11:163–200, 1987.

[Luria, 1987] M. Luria. Expressing concern. In *Proc. of the 25th Conf. of the Association for Computational Linguistics*, pages 221–227, Stanford, Calif., 1987.

[McKeown *et al.*, 1985] K. R. McKeown, M. Wish, and K. Matthews. Tailoring explanations for the user. In *Proc. of the Ninth Int. Joint Conf. on Artificial Intelligence*, pages 794–798, Los Angeles, Calif., 1985.

[Neufeld, 1989] E. Neufeld. Defaults and probabilities; extensions and coherence. In *Proc. of the First Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 312–323, Toronto, Ont., 1989.

[Pollack, 1984] M. E. Pollack. Good answers to bad questions: Goal inference in expert advice-giving. In *Proc. of the Fifth Canadian Conf. on Artificial Intelligence*, pages 20–24, London, Ont., 1984.

[Pollack, 1986] M. E. Pollack. *Inferring Domain Plans in Question-Answering.* PhD thesis, Univ. of Pennsylvania, 1986. Available as: SRI International Technical Note 403.

[Poole, 1989] D. L. Poole. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5:97–110, 1989.

[Quilici *et al.*, 1988] A. Quilici, M. G. Dyer, and M. Flowers. Recognizing and responding to plan-oriented misconceptions. *Computational Linguistics*, 14:38–51, 1988.

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

[van Beek, 1987] P. van Beek. A model for generating better explanations. In *Proc. of the 25th Conf. of the Association for Computational Linguistics*, pages 215–220, Stanford, Calif., 1987.

[Vilain, 1990] M. Vilain. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proc. of the Eighth National Conf. on Artificial Intelligence*, pages 190–197, Boston, Mass., 1990.