Temporal Reasoning during Plan Recognition

Fei Song and Robin Cohen
Dept. of Computer Science, Univ. of Waterloo
Waterloo, Ontario, Canada N2L 3G1
{fsong,rcohen}@watdragon.uwaterloo.ca

Abstract

This paper presents a strengthened algorithm for temporal reasoning during plan recognition, which improves on a straightforward application of Allen's reasoning algorithm. This is made possible by viewing plans as both hierarchical structures and temporal networks. As a result, we can show how to use as constraints the temporal relations explicitly given in input to improve the results of plan recognition. We also discuss how to combine the given constraints with those prestored in the system's plan library to make more specific the temporal constraints indicated in the plans being recognized.

Introduction

Plan recognition is the process of inferring an agent's plan based on the observation of the agent's actions. A recognized plan is useful in that it allows us to decide an agent's goal as well as predict the agent's next action. Suppose we observe that John has made the sauce and he is now boiling the noodles. Then, based on the plan in figure 1, we can decide that John's goal is to make a pasta dish and predict that his next action is to put noodles and sauce together.

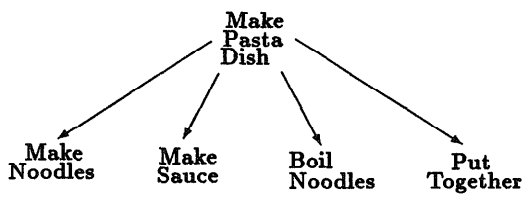

Figure 1: Hierarchical Structure of a Plan

Plan recognition has found applications in many research areas such as story understanding ([Schank and Abelson, 1977], [Bruce, 1981], [Wilensky, 1983]), psychological modeling [Schmidt *et al.*, 1978], natural language pragmatics ([Allen, 1983b], [Litman, 1985], [Carberry, 1986]), and intelligent interfaces ([Huff and Lesser, 1982], [Goodman and Litman, 1990]).

Most plan recognition models assume a library of typical plans that can occur in a particular domain. Then, a search and matching mechanism is used to recognize all the plans that contain the observed actions, called candidate plans. One problem is that it is difficult to unambiguously decide the plan of an agent, since the observation of the agent's actions is often incomplete and some actions may appear in many different plans of the system's plan library. Kautz [1987] suggests that one way of reducing the set of candidate plans is to use the temporal relations explicitly given in the observations as constraints to eliminate the plans that are inconsistent with the given constraints[1]. However, Kautz only provided a simplified procedure for checking temporal constraints and did not elaborate on the types of temporal constraints that are necessary to be represented in the system's plan library.

In this paper, we assume a model for plan recognition that is similar to Allen's [1983b], with the focus being on temporal reasoning, the process of checking the inconsistencies between the temporal constraints given in the input and those prestored in the plans of the system's plan library. Allen [1983a] proposed an algorithm that can be used to perform this task. However, Allen's algorithm [Allen, 1983a] can give weak results when applied to plan recognition, specifically for the case where some actions are defined in terms of their decomposed subactions. Our contribution is to provide a closing procedure, which makes specific the temporal constraints between an action and its decomposed subactions and works interactively with Allen's algorithm to obtain strengthened results. Moreover, we discuss briefly that in a natural language setting, the process of deriving the temporal constraints from the input through linguistic analysis, called temporal analysis, can benefit from temporal reasoning as well.

Two Different Views of Plans

A plan can be viewed as a hierarchical structure, organized by the decomposition of an action into its subactions. In the cooking plan introduced earlier, MakePastaDish is an action that can be decomposed into sub-

[1]Other solutions include the use of preference heuristics ([Allen, 1983b], [Litman, 1985], [Carberry, 1986]) and probabilities [Goldman and Charniak, 1988].

actions: MakeNoodles, MakeSauce, BoilNoodles, and PutTogether.

A plan can also be viewed as a temporal network, indicating the temporal constraints that must hold between the intervals of all actions and states in the plan. We can use Allen's interval algebra [Allen, 1983a] to represent the temporal constraints between intervals. Given intervals X and Y, there can be thirteen basic relations between them, as shown in the following table. Constraints that are less certain than basic relations are represented as the disjunction of a set of basic relations. For convenience, we define two high level constraints as follows: "Precedes" = {b, m, o} and "Includes" = {si, di, fi,eq}.

Primitives	*Symbols*	*Inverses*	*Examples*
X Before Y	X b Y	Y bi X	XXX YYY
X Meets Y	X m Y	Y mi X	XXXYYY
X Overlaps Y	X o Y	Y oi X	XXX YYY
X Starts Y	X s Y	Y si X	XXX YYYYY
X During Y	X d Y	Y di X	XXX YYYYY
X Finishes Y	X f Y	Y fi X	XXX YYYYY
X Equal Y	X eq Y	Y eq X	XXX YYY

Roughly, "Includes" describes the decomposition of an action into its subactions, since the interval of the action includes all the intervals of the subactions, while "Precedes" holds when one action "enables" another action, as the enabling action must be executed prior to the enabled action[2]. As a result, the temporal network for the cooking plan can be given in figure 2. Relations that are more specific than "Precedes" and "Includes" can be indicated as shown.

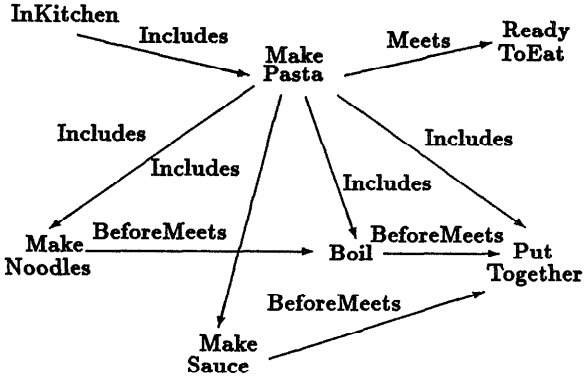

Figure 2: Temporal Structure of a Plan

[2]A definition of enablement can be found in [Pollack, 1986]. For example, finding a phone number enables the action of making a phone call.

A hierarchical structure provides a straightforward view showing all the actions in a plan, but it does not exhibit the states and temporal relations between actions. A temporal network, on the other hand, gives a detailed representation that allows all the actions, states, and their relationships to be clearly specified. However, since a temporal network treats all the intervals as the same, the temporal dependency between an action and its subactions is not explicitly indicated. We will argue that both views of a plan are important for doing temporal reasoning during plan recognition.

Algorithms for Temporal Reasoning

Weak Results of Allen's Algorithm

Allen [1983a] proposed a reasoning algorithm that propagates a new constraint to others and at the same time checks for inconsistencies. To reason about constraints, two operations of intersection and composition are defined. Intersection is just set intersection between two constraints. Composition of two constraints is the set of pair-wise multiplications between all the basic relations in the two constraints, i.e.,

$$C1 \circ C2 = \{a \times b \mid a \in C1 \text{ and } b \in C2\},$$

where the result of $a \times b$ can be looked up in a predefined table [Allen, 1983a]. For example, given C1 = {b, m, o} and C2 = {m, o}, the composition of C1 ∘ C2 is {b, m, o}.

However, Allen's algorithm can provide weak results when applied to plans that contain decompositions. Consider an example of one decomposition shown in figure 3. Here, we use A to denote the interval of the action, and a1 and a2, the intervals of the two subactions. Suppose that in the initial specification of the plan library there is no constraint between a1 and a2. Then, all we can decide are: A {si, di, fi, eq} a1 and A {si, di, fi, eq} a2.

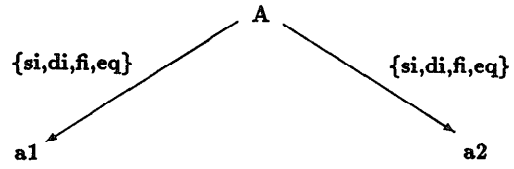

Figure 3: An Example of One Decomposition

Now, assume that from the input we get a new constraint of {b} between a1 and a2. Then, based on Allen's algorithm, we are able to propagate it to the other constraints and obtain the results shown in figure 4 (a). However, if we know that a1 and a2 are the only two subintervals of the interval A and that a1 {b} a2, then we should be able to decide A {si} a1 and A {fi} a2, i.e., a1 and a2 together comprise all of A. These desired results are shown in figure 4 (b).

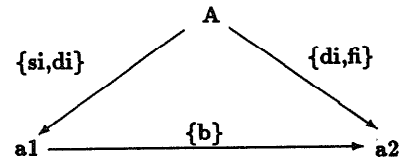

Figure 4 (a): Weak Results from Allen's Algorithm

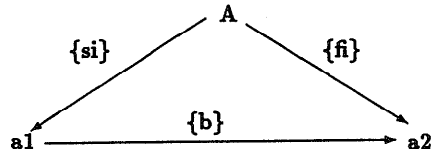

Figure 4 (b): Strong Results Desired

These weak results can be carried further when we consider a network that consists of more than one decomposition. Suppose that at the beginning we have the network shown in figure 5, where "inc" stands for {si, di, fi, eq}.

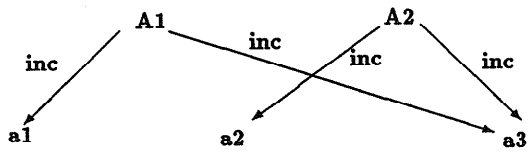

Figure 5: An Example of Two Decompositions

When the constraints: a1 {b} a2 and a2 {b} a3 are given from the input, we can apply Allen's algorithm to obtain the results shown in figure 6 (a). Here, "com" stands for the constraint: {o, oi, s, si, d, di, f, fi, eq}. However, using a similar argument as made in the previous example, we should get the stronger results shown in figure 6 (b).

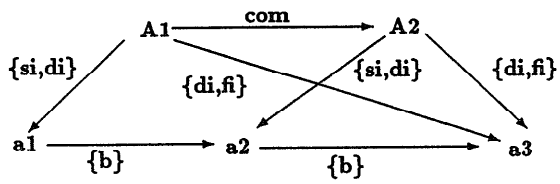

Figure 6 (a): Weak Results from Allen's Algorithm

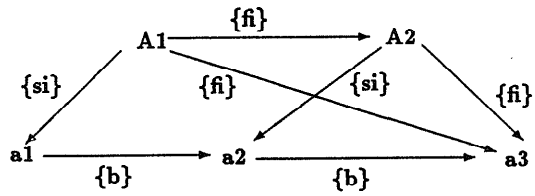

Figure 6 (b): Strong Results Desired

There are also networks that are considered as consistent by Allen's algorithm but in fact are not when used to represent decompositions. For instance, if the constraint between A and a2 in figure 4 (b) is labeled as {di}, Allen's algorithm would decide this as consistent, but as we argued above, this is in fact not consistent with the desired results.

The reason that Allen's algorithm gives us weak results when applied in plan recognition is that it treats all the intervals of actions as independent of each other. In plans where actions are connected through decompositions, the intervals of abstract actions actually depend on the intervals of their decomposed subactions. To make these dependencies explicit in the reasoning process, we must acknowledge that decompositions of abstract actions into their subactions are complete; no more subactions are needed for or can be added to the decompositions. This will allow us to compute the boundary intervals of the abstract actions based on all the constraints between the subactions. For instance, if there is a linear ordering between all the subactions, we will be able to decide that the abstract action is temporally bounded by the subactions that occur the earliest and the latest. We say that a decomposition is closed if the interval of the abstract action is temporally bounded by the intervals of the subactions.

Closing Procedures for Decompositions

We can classify the 13 basic temporal relations introduced in section 2 into five classes: {b, m, o}, {bi, mi, oi}, {si, di, fi}, {s, d, f}, and {eq}. Then, we can divide a constraint into five subsets accordingly. Given the constraint {o, oi, s, si, d, di, f, fi, eq}, for example, the corresponding five subsets are: {o}, {oi}, {si, di, fi}, {s, d, f}, and {eq}. Let C be the constraint between the two subactions of a decomposition. Then, for each non-empty subset of C, we can provide a simple solution to close the decomposition, as shown in figure 7.

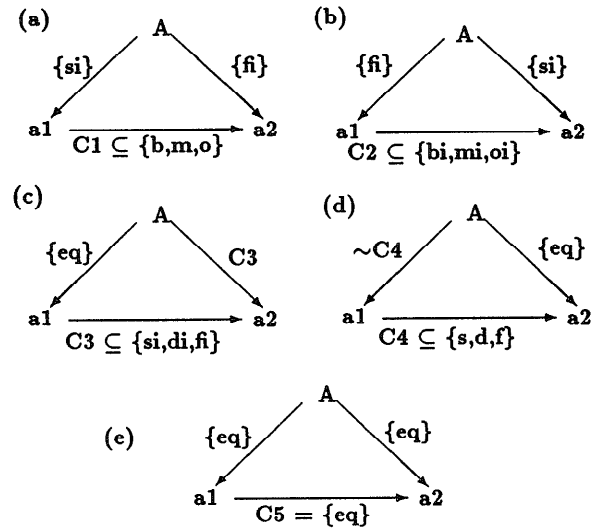

Figure 7: Five Special Cases for Closing a Decomposition

Case (a) indicates that A is bounded by a1 and a2 since for any subset of C1={b, m, o} the start part of a1 is clearly located before the end part of a2. Case (c) suggests that A is bounded by a1 since for any subset

of C3={si, di, fi} we can decide that A {eq} a1, and from the composition of A {eq} a1 and a1 C3 a2 we can derive A C3 a2. Cases (b) and (d) are the reverse cases of (a) and (c), and case (e) is trivially justified.

If only one subset of C is not empty, then the corresponding case above already provides a solution for closing the decomposition. However, a constraint generally has more than one non-empty subset. As a result, the solution of closing a decomposition should be the disjunction of all the cases that contain these non-empty subsets. We can illustrate the closing process by viewing a decomposition as the conjunction of all its constraints. For example, the formula for a decomposition of two subactions may be described as:

A {si, di, fi, eq} a1 ∧ A {si, di, fi, eq} a2 ∧
a1 {b, bi} a2

By dividing {b, bi} into two subsets, {b} and {bi}, we can change it into an equivalent formula:

(A {si, di, fi, eq} a1 ∧ A {si, di, fi, eq} a2 ∧
a1 {b} a2) ∨
(A {si, di, fi, eq} a1 ∧ A {si, di, fi, eq} a2 ∧
a1 {bi} a2)

Now it becomes clear that each subformula above corresponds to a special case in figure 7. So we can close both subformulas and get the disjunction of two special cases:

(A {si} a1 ∧ A {fi} a2 ∧ a1 {b} a2) ∨
/* from case (a) */
(A {fi} a1 ∧ A {si} a2 ∧ a1 {bi} a2)
/* from case (b) */

Unfortunately, this result is too strong in that we have to divide the temporal network of a plan into two networks, each containing a special case. For a plan that consists of many decompositions, the total number of different networks could be much larger. In order to retain just one network while still taking decompositions into account, we will have to relax our result to some extent. Here, we merge the two subformulas by taking the disjunctions of the corresponding constraints:

A {si, fi} a1 ∧ A {si, fi} a2 ∧ a1 {b, bi} a2

This formula is equivalent to the previous result, but it also contains two redundant, inconsistent cases. This can be seen from the expansion of the formula:

(A {si} a1 ∧ A {fi} a2 ∧ a1 {b} a2) ∨
(A {fi} a1 ∧ A {si} a2 ∧ a1 {bi} a2) ∨
(A {si} a1 ∧ A {si} a2 ∧ a1 {b} a2) ∨
/* inconsistent */
(A {fi} a1 ∧ A {fi} a2 ∧ a1 {bi} a2)
/* inconsistent */

However, even though our result is weakened, it is still stronger than that from Allen's algorithm in most cases. For the example above, the result from Allen's algorithm will be:

A {si, di, fi} a1 ∧ A {si, di, fi} a2 ∧ a1 {b, bi} a2

In the following, we summarize our discussion and provide a procedure for closing a decomposition of two subactions. Here, R(k, n) is a relation given between two intervals labeled as nodes k and n.

```
function close-two(R(k, n))
begin
    create a dummy node labeled temp;
    R(temp, k) <- { };
    R(temp, n) <- { };
    if R(k, n) ∩ {b, m, o} is not empty then
        R(temp, k) <- R(temp, k) ∪ {si};
        R(temp, n) <- R(temp, n) ∪ {fi};
    if R(k, n) ∩ {bi, mi, oi} is not empty then
        R(temp, k) <- R(temp, k) ∪ {fi};
        R(temp, n) <- R(temp, n) ∪ {si};
    C3 <- R(k, n) ∩ {si, di, fi};
    if C3 is not empty then
        R(temp, k) <- R(temp, k) ∪ {eq};
        R(temp, n) <- R(temp, n) ∪ C3;
    C4 <- R(k, n) ∩ {s, d, f};
    if C4 is not empty then
        R(temp, k) <- R(temp, k) ∪ ~C4;
        R(temp, n) <- R(temp, n) ∪ {eq};
    if R(k, n) ∩ {eq} is not empty then
        R(temp, k) <- R(temp, k) ∪ {eq};
        R(temp, n) <- R(temp, n) ∪ {eq};
    return temp
end
```

Having developed the procedure of "close-two", we are now in a position to extend the result to close a decomposition of more than two subactions. Figure 8 (a) shows a decomposition of three subactions.

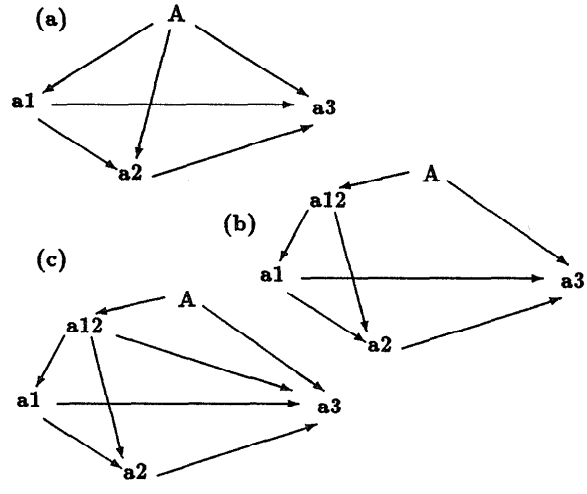

Figure 8: Closing a Decomposition of Three Subactions

In order to close the decomposition, we introduce an intermediate action a12 that takes a1 and a2 as subactions, shown in (b). Now, for this new decomposition, we can call "close-two" and get the closed constraints between a12 and a1 and between a12 and a2. Then,

based on these results, we can compute a new constraint between a12 and a3, shown in (c). Now, a12 and a3 form the two subactions of A. Once again, we can call "close-two" and get the closed constraints between A and a12 and between A and a3. At this time, the constraint between A and a3 has been closed. To get the closed constraints between A and a1 and between A and a2, we can perform the compositions of A to a1 and a2 via a12. Since all the constraints have been closed, we can eliminate the intermediate action a12 and all the constraints connected to it. The result brings us back to the structure in (a), but this time, all the constraints from A to its subactions have been closed. The above process can be repeated if there are more subactions to be closed.

We can now give a general procedure, which closes a decomposition of any number of subactions by using our "close-two" procedure. Here, k denotes an abstract action, and S, a list of the subactions of the abstract action. Also, given nodes i and j, N(i, j) corresponds to the existing constraint, and R(i, j), a new or derived constraint between i and j. Finally, U denotes the disjunctive set of all possible primitive interval relations, i.e., U = {b, bi, m, mi, o, oi, s, si, d, di, f, fi, eq}.

```
procedure close-all(k, S)
begin
    get first n from the list S;
    N(k, n) <- {eq};
    C <- {n};

    while S is not empty do begin
        get next n from the list S;
        R(k, n) <- U;
        foreach c in C do
            R(k, n) <- R(k, n) ∩ N(k, c) ∘ N(c, n);
        temp <- close-two(R(k,n));
        foreach c in C do
            N(k, c) <- R(temp, k) ∘ N(k, c);
        N(k, n) <- R(temp, n);
        C <- C ∪ {n}
    end
end
```

Our Strengthened Algorithm

Our closing procedure is built on the temporal constraints between all the subactions. In order to get stronger results, we first view plans as temporal networks and use Allen's algorithm to make these constraints more specific. Then, we view plans as hierarchical structures and close all the decompositions in a depth first order. Once all the decompositions are closed, some of the constraints in the network may be updated. As a result, we need to call Allen's algorithm again to propagate these constraints. In general, we can design a strengthened algorithm by interactively calling Allen's algorithm and our closing procedure a number of times. Such a process will eventually terminate since every time we update a constraint, some of its basic relations will be eliminated and there are at most 13 basic relations in a constraint[3].

Applications of Temporal Reasoning

There are two possible results that can be obtained from temporal reasoning: if the given constraints are inconsistent with the prestored constraints of a candidate plan, then the plan will be eliminated; otherwise, the given constraints will be added to make the prestored constraints more specific.

Here is an example to show the importance of doing temporal reasoning during plan recognition. Suppose that our plan library contains two plans for making GuoTie and JianJiao, two common ways of making fried dumplings in Chinese cooking, shown in figure 9.

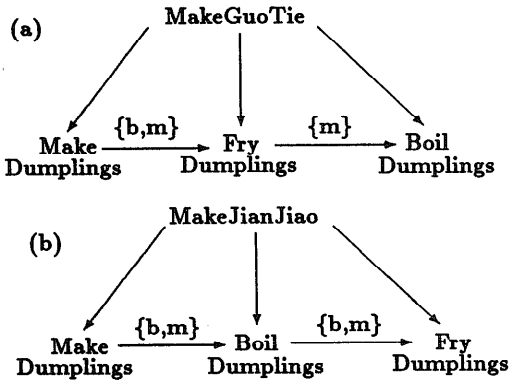

Figure 9: Two Simplified Plans in a Plan Library

Then, given the observation that BoilDumplings occurs earlier than FryDumplings[4], a plan recognition model that does not use temporal constraints from the input would propose MakeGuoTie and MakeJianJiao as the candidate plans, for both of them contain the two given actions. However, by taking the temporal relations given in the input as a constraint and checking them with those prestored in candidate plans, we find that MakeGuoTie is inconsistent with the given constraint, as BoilDumplings occurs later than FryDumplings in this plan. As a result, our plan recognition model will only propose MakeJianJiao as the plan that the agent is performing.

The other result of making prestored constraints more specific can benefit the process of deriving the temporal constraints from observation, which we call "temporal analysis." In a natural language setting, the need for automating temporal analysis becomes important, as the observations are described in terms of

[3]Due to the space limitation, the algorithm is not given here, but the readers should be able to construct it easily based on the discussion of this section.

[4]In a natural language setting, for example, such a temporal constraint may be obtained by linguistically analyzing the input: "I have boiled the dumplings and am now frying them."

utterances and the temporal constraints suggested by linguistic expressions such as tense, aspect, temporal adverbials and connectives. However, as pointed out in ([Webber, 1988], [Allen, 1988], [Song, 1990]), these expressions are sometimes not strong enough to help derive specific temporal relations between the actions mentioned in the input; other discourse indicators such as cue-phrases and world knowledge are also needed for doing temporal analysis. Temporal reasoning is useful in that it provides a way of combining the given constraints and the prestored constraints in a candidate plan. In cases where the constraints indicated in the input are specific, we can use them to reduce the set of candidate plans, but in cases where the given constraints are vague, the prestored constraints in a candidate plan can be used to fill in the details about the temporal relations between actions. Readers are referred to Song [1990] for more discussion on temporal analysis.

Conclusion

In this paper, we present a strengthened algorithm for temporal reasoning during plan recognition. We view plans as both hierarchical structures and temporal networks. This allows us to design a closing procedure which makes specific the temporal constraints between an action and its decomposed subactions and works interactively with Allen's algorithm to obtain strengthened results. Two main applications of temporal reasoning are to reduce the number of candidate plans during plan recognition and to help derive the temporal constraints from natural language input through linguistic analysis.

Note that the strengthened algorithm is not quite efficient in that it has to call our closing procedure and Allen's algorithm interactively. Some results on localizing the propagation in Allen's algorithm have been reported in [Koomen, 1989]. Future work should be directed to find efficient ways for combining the two processes during temporal reasoning.

Acknowledgements

We would like to thank Peter van Beek and the anonymous referees for their useful comments. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Institute of Computer Research (ICR), and the University of Waterloo.

References

Allen, James F. 1983a. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Allen, James F. 1983b. Recognizing intentions from natural language utterances. In Brady, M. and Berwick, R., editors 1983b, *Computational Models of Discourse*. The MIT Press. 107–166.

Allen, James F. 1988. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company.

Bruce, B. 1981. Plans and social actions. In Spiro, R.; Bruce, B.; and Brewer, W., editors 1981, *Theoretical Issues in Reading Comprehension*. Lawrence Erlbaum, Hillsdale.

Carberry, Sandra 1986. *Pragmatic Modeling in Information System Interfaces*. Ph.D. Dissertation, University of Deleware.

Goldman, Robert and Charniak, Eugene 1988. A probabilistic ATMS for plan recognition. In *AAAI-88 Workshop on Plan Recognition*.

Goodman, Bradley A. and Litman, Diane J. 1990. Plan recognition for intelligent interfaces. In *IEEE Conference on Artificial Intelligence Applications*.

Huff, Karen and Lesser, Victor 1982. Knowledge-based command understanding: An example for the software development environment. Technical Report TR 82-6, Computer and Information Science, University of Massachusetts, Amherst.

Kautz, Henry A. 1987. *A Formal Theory of Plan Recognition*. Ph.D. Dissertation, University of Rochester.

Koomen, Johannes A. 1989. Localizing temporal constraint propagation. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Canada. 198–202.

Litman, Diane 1985. *Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues*. Ph.D. Dissertation, University of Rochester.

Pollack, Martha E. 1986. *Inferring Domain Plans in Question-Answering*. Ph.D. Dissertation, University of Pennsylvania.

Schank, Roger C. and Abelson, Robert P. 1977. *Scripts, Plans, Goals, and Understanding*. L. Erlbaum Associates, Hillsdale, New Jersey.

Schmidt, C. F.; Sridharan, N. S.; and Goodson, J. L. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11:45–83.

Song, Fei 1990. *A Processing Model for Temporal Analysis and its Application to Plan Recognition*. Ph.D. Dissertation, University of Waterloo.

Webber, Bonnie Lynn 1988. Tense as discourse anaphor. *Computational Linguistics* 14(2):61–73.

Wilensky, Robert 1983. *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley Publishing Company.