Journal of Artificial Intelligence Research 15 (2001) 1-30 Submitted 11/00; published 7/01

Goal Recognition through Goal Graph Analysis

Jun Hong J.HONG@ULST.AC.UK
School of Information and Software Engineering
University of Ulster at Jordanstown
Newtownabbey, Co. Antrim BT37 0QB, UK

Abstract

We present a novel approach to goal recognition based on a two-stage paradigm of graph construction and analysis. First, a graph structure called a Goal Graph is constructed to represent the observed actions, the state of the world, and the achieved goals as well as various connections between these nodes at consecutive time steps. Then, the Goal Graph is analysed at each time step to recognise those partially or fully achieved goals that are consistent with the actions observed so far. The Goal Graph analysis also reveals valid plans for the recognised goals or part of these goals.

Our approach to goal recognition does not need a plan library. It does not suffer from the problems in the acquisition and hand-coding of large plan libraries, neither does it have the problems in searching the plan space of exponential size. We describe two algorithms for Goal Graph construction and analysis in this paradigm. These algorithms are both provably sound, polynomial-time, and polynomial-space. The number of goals recognised by our algorithms is usually very small after a sequence of observed actions has been processed. Thus the sequence of observed actions is well explained by the recognised goals with little ambiguity. We have evaluated these algorithms in the UNIX domain, in which excellent performance has been achieved in terms of accuracy, efficiency, and scalability.

1. Introduction

Plan recognition involves inferring the intentions of an agent from a set of observations. A typical approach to plan recognition uses an explicit representation of possible plans and goals, often called a plan library, and conducts some type of reasoning on the basis of a set of observations to identify plans and goals from the plan library, that could have caused the observations.

Plan recognition is useful in many areas, including discourse analysis in natural language question-answering systems, story understanding, intelligent user interfaces, and multi-agent coordination. Much early research in plan recognition has been done in natural language question-answering systems (Allen & Perrault, 1980; Allen, 1983; Sidner, 1985; Litman & Allen, 1987; Carberry, 1988; Pollack, 1990; Grosz & Sidner, 1990). In these systems, plan recognition has been used to support intelligent response generation; to understand sentence fragments, ellipsis and indirect speech acts; to track a speaker's flow through a discourse; and to deal with correctness and completeness discrepancies between the knowledge of users and systems.

Plan recognition can enhance user interfaces. The recognition of the user's goals and plans from his interaction with the interface facilitates intelligent user help (Carver, Lesser, & McCue, 1984; Huff & Lesser, 1988; Goodman & Litman, 1992; Bauer & Paul, 1993; Lesh & Etzioni, 1995). Plan recognition enables the interface to assist the user in task completion, and error detection and recovery (Wilensky & et al., 1988). The interface does this by watching over the user's shoulder to infer his goals and plans. It can then decide what help and assistance the user needs.

In story understanding (Schank & Abelson, 1977; Wilensky, 1983; Charniak & Goldman, 1993), it is useful to recognise the goals and plans of the characters from the described actions in order to understand what the characters were doing. In multi-agent coordination, efficient and effective coordination among multiple agents requires modelling each agent's goals and plans (Huber, Durfee, & Wellman, 1994; Huber & Durfee, 1995).

Given a set of observations, most plan recognition systems (Allen & Perrault, 1980; Carberry, 1986; Litman & Allen, 1987; Kautz, 1987; Pollack, 1990)) search a space of possible plan hypotheses for candidate plans and goals that account for the observations. To form the search space in a given domain, some kind of plan library is required. For instance, in Kautz's event hierarchy (Kautz, 1987), plan decompositions are required that describe how low-level actions make up complex actions. Despite its obvious advantage of expressive richness, use of a plan library has a few limitations. First, it is not able to deal with new plans whose types do not appear in the plan library. Second, acquiring and hand-coding the plan library in a large and complex domain presents a tedious or impractical task. Third, in some other domains, the knowledge about plans might not be readily available.

Some attempts (Mooney, 1990; Forbes, Huang, Kanazawa, & Russell, 1995; Lesh & Etzioni, 1996; Albrecht, Zukerman, & Nicholson, 1998; Bauer, 1998) have recently been made to apply machine learning techniques to the automated acquisition and coding of plan libraries. Even when leaving aside the plan library consideration, searching the plan space can, however, be exponentially expensive because the number of possible plan hypotheses can be exponential in the number of actions (Kautz, 1987). Most plan recognition systems have been developed in domains in which people's behaviour is characterized by fewer than 100 plans and goals (Lesh & Etzioni, 1996).

In this paper, we focus on goal recognition, a special case of plan recognition. We introduce a novel approach to goal recognition, in which graph construction and analysis is used as a paradigm. Our approach significantly differs from most plan recognition systems. First, our approach does not need a plan library. Instead we define what constitutes a valid plan for a goal. We need to consider only how observed actions can be organised into plans. Hence we do not have the problems associated with the acquisition and hand-coding of the plan library in a large and complex domain as well as the availability of planning knowledge in a domain. Most plan recognition systems cannot recognise any new plans whose types do not appear in the plan library. Without using a plan library, our approach does not suffer from this limitation. Second, instead of immediately searching for a plan in the plan space as in most plan recognition systems, our approach explicitly constructs a graph structure, called a Goal Graph, which is then analysed to recognise goals and plans. Our approach therefore does not have the problems in searching the plan space of exponential size. Third, our approach recognises only partially or fully achieved goals that are consistent with the actions observed so far. The number of such recognised goals is usually very small after a sequence of observed actions has been processed. Thus the sequence of observed actions is well explained by the recognised goals with little ambiguity.

It should be emphasised that in our approach to goal recognition, the purpose of recognising partially or fully achieved goals is to explain past actions rather than predicting future actions. This is particularly useful in such problem areas as story understanding, software advisory systems, database query optimisation, and customer data mining. These problem areas have several specific characteristics. First, most actions have been either described or observed. Second, it is very likely that the user's intended goal has been partially or fully achieved by these actions. Third, recognising the intended goal aims at explaining past actions rather than predicting future actions. Finally, distinguishing partially or fully achieved goals from the others greatly reduces ambiguity involved in recognising the intended goal.

In story understanding, most actions of the characters are described in the story. Recognising goals and plans that account for the described actions enables better understanding of what the characters were doing. In software advisory systems, after a user has been observed to issue a sequence of operations in a software application, the system can first recognise the task the user has performed. The system can then decide whether the user has performed the task in a suboptimal way, and advice can be given to the user as to how to better perform the task. In database query optimisation, after a user has conducted a sequence of data retrieval and manipulation operations, recognising the underlying query can lead to advice on query optimisation when the query has not been executed in an optimal way. In customer data mining, the individual customer's shopping goals can be recognised from logged customer on-line shopping data. This can also form a basis for performing other customer data mining tasks.

Our algorithms for Goal Graph construction and analysis are both provably sound, polynomial-time, and polynomial-space. Our empirical results in the UNIX domain show that our algorithms perform well in terms of accuracy, efficiency, and scalability. They also show that our algorithms can be scaled up and applied to domains in which there are tens of thousands of possible goals and plans. Though our algorithm for Goal Graph analysis is not complete, it recognised every goal that was intended and successfully achieved by the subject in the UNIX data set we used in our evaluation. Since our new graph-based approach to goal recognition is fundamentally different from the existing methods for plan recognition, it provides an alternative to these methods and a new perspective on the plan recognition problem.

The rest of the paper is organised as follows. First, we give an overview of our novel approach to goal recognition. In Section 3, we discuss the domain representation. In Section 4 we define Goal Graphs, valid plans, and consistent goals. In Section 5 we present our goal recognition algorithms together with our analysis of these algorithms. In Section 6 we discuss the empirical results. We summarise the paper and discuss limitations and future work in the last section.

2. A Novel Approach to Goal Recognition

In this section, we describe the basic assumptions we make on the goal recognition problem and outline our approach. We discuss previous work on planning with Planning Graphs and a graph-based approach to goal recognition. We briefly describe empirical results that are in favour of our approach.

2.1 Basic Assumptions

We start with an example in the UNIX domain. When we observe that a user types two commands, `cd papers` and `ls`, one after another, we should be able to infer that the user wants to find a file or subdirectory in the directory, `papers`, for two reasons. First, this goal has been fully achieved. Second, it is relevant to both commands in a consistent way in the sense that the first command satisfies one of the preconditions of the second command and the second command achieves the recognised goal. The recognised goal might be just an intermediate goal of the user. The user's intended goal might be one of the file related goals, for instance, deleting a file from the directory. Since these commands can both be part of plans for almost all the file related goals, it is impossible for us to uniquely identify the user's intended goal at the current time step. Yet the goal of finding a file or directory well explains these commands. If the user next types a command, `rm oldpaper.tex`, we can then infer that the user's goal is to delete a file, `oldpaper.tex`, in the directory, `papers`, because this goal is now fully achieved and it is relevant to all the commands observed so far in a consistent way in the sense that the first command satisfies one of the preconditions of the second and third commands, the second command satisfies one of the preconditions of the third command, and the third command achieves the recognised goal.

This example highlights the way our new approach to goal recognition works. We make the following assumptions on the goal recognition problem. First, a set of actions are observed at consecutive time steps.[1] Second, the initial state of the world immediately before the set of actions are observed is known.[2] Third, we have domain knowledge on actions and goals, that is, we know the preconditions and effects of every observed action, and every possible goal is explicitly specified by a set of goal descriptions.

Given these assumptions, when an action is observed at a time step, we want to infer which goals have been partially or fully achieved at this time step and whether each of the achieved goals is relevant to the strict majority of the actions observed so far in a consistent way in the sense that these actions can be organised into a plan structure for the goal or part of it.

2.2 Goal Recognition with Goal Graph

We propose to use a graph structure, called a Goal Graph, in our new approach to goal recognition. We view the goal recognition problem as a process of graph construction and analysis. In a Goal Graph, action nodes represent the actions observed at consecutive time steps; proposition nodes represent the state of the world at consecutive time steps, as it is changed from the initial state to the subsequent states by the observed actions; and goal nodes represent the goals that are partially or fully achieved at consecutive time steps. Edges in a Goal Graph explicitly represent relations between actions and propositions as well as relations between propositions and goals. Based on these explicit relations in the constructed Goal Graph, causal links between either two actions or an action and a goal can be recognised. Having recognised these causal links, it can be decided whether a fully or partially achieved goal at a time step is relevant to the strict majority of the observed

1. The observed actions can be partially ordered in the sense that more than one action can be observed at a time step and there is no temporal ordering constraint on these actions.
2. Our approach however reasons about the state of the world at subsequent time steps.

actions so far in a consistent way in the sense that these relevant actions can be organised into a plan structure for the goal or part of it. In our approach, extraneous, redundant, and partially ordered actions in plans are all handled.

Our attempt to use graph construction and analysis as a paradigm for goal recognition is in spirit influenced by Blum and Furst's efforts on planning with Planning Graphs (Blum & Furst, 1997). They introduced a new graph-based approach to planning in STRIPS domains, in which a graph structure called a Planning Graph is first constructed explicitly rather than searching immediately for a plan as in standard planning methods. Many useful constraints inherent in the planning problem are made explicitly available in a Planning Graph to reduce the amount of search needed. The Planning Graph is then analysed to generate possible plans.

Our Goal-Graph-based approach to goal recognition can be seen as a counterpart of planning with Planning Graph. Though graph structures are used in both approaches, they are composed of different kinds of nodes and edges. In a time step, a Planning Graph represents all the possible propositions either added by the actions in the previous time step or brought forward by maintenance actions from the previous time step, and all the possible actions whose preconditions have been satisfied by the propositions in the time step. On the other hand, a Goal Graph, in a time step, represents only the propositions either added by the actions observed in the previous time step or brought forward by maintenance actions from the previous time step, and the actions observed in the time step. In addition, a Goal Graph, in a time step, also represents all the possible goals, either fully or partially achieved in the time step, while a Planning Graph does not represent any goal at all. Accordingly, a Planning Graph represents only relations between actions and propositions, while a Goal Graph also represents relations between propositions and goals.

The analysis of a Planning Graph aims to search for all the possible subgraphs of the Planning Graph, that form valid plans for the only given goal. On the other hand, the analysis of a Goal Graph aims to search for every possible partially or fully goal for which there exists a subgraph of the Goal Graph, consisting of the strict majority of the observed actions. Such a subgraph forms a valid plan for the goal or part of it and shows that the strict majority of the observed actions are relevant to the goal in a consistent way.

The domain representation can be the same for both planning with Planning Graph and goal recognition with Goal Graph. In this regard, previous efforts on handling more expressive representation languages (Gazen & Knoblock, 1997; Anderson, Smith, & Weld, 1998; Koehler, Nebel, Hoffmann, & Dimopoulos, 1997) are still useful for goal recognition. These languages allow use of disjunctive preconditions, conditional effects, and universally quantified preconditions (goal descriptions) and effects in action and goal representation. Our ADL-like domain representation is actually based on this work, that allows use of conditional effects, universally quantified effects, and existentially and universally quantified preconditions and goal descriptions in the action and goal representation.

Our Goal-Graph-based approach further extends Lesh and Etzioni's previous work on use of a graph representation of actions and goals for the goal recognition problem (Lesh & Etzioni, 1995). They used a graph representation, called a consistency graph, for the goal recognition problem. A consistency graph consists of action and goal nodes representing possible actions and goals, and edges representing possible connections between nodes in the graph. Initially, action and goal nodes are fully connected to each other in a consistency graph, and inconsistent goals are then repeatedly pruned from the consistency graph.

There are a number of major differences between Lesh and Etzioni's approach and ours. First, two different graph representations are used. Apart from action and goal nodes, a consistency graph does not have nodes representing the propositions that model how the state of the world is changed by the observed actions. Therefore, the consistency graph does not explicitly reveal causal links over actions and goals. Neither does their system know whether a goal has been partially or fully achieved by the observed actions. Our Goal Graph consists of action, goal, and proposition nodes. It explicitly reveals causal links over actions and goals, hence our system knows how the observed actions are composed into valid plans for the recognised goals or part of these goals. Our systems also knows whether a goal is partially or fully achieved by the observed actions.

Second, goal consistency is defined differently. In Lesh and Etzioni's approach, a goal is consistent if there exists a plan that includes the observed actions and achieves the goal. In our approach, a goal is consistent if it has been partially or fully achieved by the observed actions and is relevant to the strict majority of the observed actions. Also, two different recognition processes are used. In their approach a pruning process is used to prune inconsistent goals from the consistency graph. The pruning process guarantees that the goals pruned from the consistency graph are inconsistent goals. However, the number of consistent goals, still remaining in the consistency graph after pruning, is usually large. Thus ambiguity on the intended goal remains an issue to be addressed. Our approach instead uses a graph analysis process to directly recognise consistent goals from only those fully or partially achieved goals. The number of consistent goals recognised in the Goal Graph is usually small. Third, their approach requires that every observed action is relevant to the goal, while only the strict majority of the observed actions are required to be relevant to the goal in our approach.

We have developed two algorithms, GoalGraphConstructor and GoalGraphAnalyser, based on our two-stage paradigm of Goal Graph construction and analysis. The GoalGraphConstructor algorithm takes a set of actions as they are observed at different time steps and constructs a Goal Graph. The GoalGraphAnalyser algorithm analyses the constructed Goal Graph to recognise consistent goals and valid plans. We prove that our algorithms are sound, polynomial-time, and polynomial-space.

Our algorithms have been implemented in Prolog and tested in the UNIX domain on a desktop with a Pentium III processor at 600 MHz. We used a set of data, collected in the UNIX domain at the University of Washington, with a domain representation of 35 action schemata and 249 goal schemata. On the entire UNIX data set, on average it took just a few CPU seconds to update the Goal Graph when an observed action was processed, and usually only a very small number of consistent goals remained after a sequence of actions had been observed. In all those test cases where the intended goals had been successfully achieved by the subjects, these intended goals were all among the remaining goals recognised after complete sequences of actions had been observed. To test the scalability of our algorithms, we tested them on a series of spaces of approximate 10^4, 2×10^4, up to 10^5 candidate goals respectively in the UNIX domain, where the approximate linear time performance has been achieved. Our empirical results show that our algorithms can be scaled up and applied to domains in which there are tens of thousands of possible goals and plans.

3. The Domain Representation

We use an ADL-like representation (Pednault, 1989), including actions with conditional and universally quantified effects, and existentially as well as universally quantified preconditions and goal descriptions. In our approach to goal recognition, a goal recognition problem consists of

- A set of action schemata specifying primitive actions.
- A finite, dynamic universe of typed objects where objects can be either added or deleted by an action.
- A set of propositions called the Initial Conditions.
- A set of goal schemata specifying possible goals.
- A set of actions that are observed at consecutive time steps.[3]

The solution to a goal recognition problem consists of a set of partially or fully achieved goals that are consistent with the set of observed actions together with the valid plans consisting of observed actions for the recognised goals or part of them.

The goal schema consists of a set of goal descriptions (GDs) that are defined by the following EBNF definitions.

```
<GD> ::= <term>
<GD> ::= (not <term>)
<GD> ::= (neg <term>)
<GD> ::= (and <GD>*)
<GD> ::= (imply <GD> <GD>)
<GD> ::= (exist <term> <GD>)
<GD> ::= (forall <term> <GD>)
<GD> ::= (eq <argument> <argument>)
<GD> ::= (neq <argument> <argument>)
```

The action schema consists of a set of preconditions and a set of effects. The set of preconditions are defined as the same as goal descriptions. The set of effects are defined by the following EBNF definitions.

```
<effect> ::= <term>
<effect> ::= (neg <term>)
<effect> ::= (and <effect>*)
<effect> ::= (when <GD> <effect>)
<effect> ::= (forall <term> <effect>)
```

3. When we say an observed action, we mean an action that has been observed and successfully executed. We ignore those invalid actions. In the UNIX domain, for instance, these invalid actions are those issued commands that UNIX failed to execute and responded only in error messages.

In the above two sets of EBNF definitions, a `<term>` is an atomic expression of the form:

```
<term> ::= (<predicate-name> <argument>*)
<argument> ::= <constant-name>
<argument> ::= <variable-name>
```

We use `eq` and `neq` to specify equality and inequality constraints. We have two negation connectives: `neg` and `not`. We use `(neg A)` to specifically mean that the truth value of `A` is made explicitly known to be false by an action. We use `(not A)` to mean that the truth value of `A` is known to be false either explicitly or implicitly. The latter kind of representation can be used when it is not necessary to represent that the truth value of `A` is explicitly known to be false as long as it is known to be false. The closed world assumption is therefore implemented as follows. In the initial state of the world, we explicitly represent only the propositions known to be true in the Initial Conditions. Any proposition not explicitly represented in the state of the world is implicitly known to be false. Actions however may add some propositions explicitly known to be false to the state of the world. A proposition can become explicitly known to be false only if it has been made explicitly known to be false by an action. It is important to represent the propositions that are explicitly known to be false, because we want to explicitly represent the effects of actions so that causal links between either two actions or an action and a goal can be established.

Both goal and action schemata are parameterised by typed variables that are represented by terms with object type predicates. A goal is a ground instance of a goal schema. An action is a ground instance of an action schema. The set of goal descriptions for a goal must be satisfied in the state of the world when the goal is fully achieved. If some but not all the goal descriptions are satisfied instead, the goal is partially achieved. Positive literals in the goal descriptions represent propositions true in the state of the world. Negative literals in the goal descriptions represent propositions known to be false in the state of the world. We use `imply` to specify dependency constraints on goal descriptions. If a goal description $\texttt{GD}_2$ is implied by another goal description $\texttt{GD}_1$, $\texttt{GD}_2$ can only be satisfied when $\texttt{GD}_1$ is satisfied but $\texttt{GD}_1$ can be satisfied without $\texttt{GD}_2$ being satisfied. A goal description can be existentially and universally quantified over a dynamic universe of objects.

The set of preconditions must be satisfied in the state of the world before the action can be executed. The set of preconditions have the same syntax and semantics as the set of goal descriptions. The set of effects are taken in the state of the world when the action is executed. Positive literals in the effects represent propositions true in the state of the world after the action is executed. These propositions are added to the state of the world. Negative literals in the effects represent propositions no longer true in the state of the world after the action is executed. These propositions are deleted from the state of the world, while the negations of these propositions are added to the state of the world, representing that these propositions are explicitly known to be false in the state of the world after the action is executed. Furthermore, a conditional effect consists of an antecedent and a consequent, where the antecedent is a set of preconditions and the consequent is a set of effects. The effects in the consequent can be taken only when the preconditions in the antecedent are satisfied in the state of the world before the action is executed. An effect in an action schema can be universally quantified over a dynamic universe of objects.

We use a simple example domain extended from Pednault's famous example (Pednault, 1988). It involves transportation of two physical objects, a dictionary, and a chequebook, between home and office using a briefcase. We assume that only one physical object can be carried in the briefcase at a time. The extended briefcase domain consists of

- A special physical object: a briefcase.
- Two physical objects: a dictionary and a chequebook.
- Two locations: home and office.
- Three action schemata:

  '*Moving the briefcase from one location to another*',
  '*Putting a physical object in the briefcase*', and
  '*Taking out a physical object from the briefcase*'.

- Three goal schemata:

  '*Moving a physical object from one location to another*',
  '*Keeping a physical object at a location*', and
  '*Keeping a physical object in the briefcase*'.

The action and goal schemata of this example domain are shown in Figure 1. They are used throughout the paper.

In the actual implementation of our goal recognition algorithms, universally quantified preconditions and effects, and conditional effects in action schemata are eliminated; and equivalent schemata are created. We use a particular approach we call *dynamic expansion.* Dynamic expansion involves two steps. In the first step, universally quantified preconditions and effects in an action schema are dynamically compiled into the corresponding Herbrand bases, taking into account the universe of objects at the current time step. These universally quantified preconditions and effects can only be dynamically compiled because we assume that the universe of objects can be dynamically changed. This assumption is needed in a domain like the UNIX shell system where destruction and creation of objects are required. Under our assumption on the dynamic universe of objects, for each object in the universe, its object type must have been declared at the time step immediately before the action is executed. For each object in the initial universe of objects, its type must be declared in the Initial Conditions. An object can be either added to or deleted from the universe of objects by an action at a time step, with an effect either stating a proposition about the new object or negating a proposition about the existing object.

For instance, suppose that at the time step immediately before an instance of action schema `mov-b` shown in Figure 1 is executed, the universe of objects consists of three physical objects: `B`, `C`, and `D`. Action schema `mov-b` is then dynamically compiled into action schema `mov-b-1` as follows.

```
(:action mov-b
   :paras (?l ?m - loc)
   :pre (and (neq ?l ?m)(at B ?l))
   :eff (and (at B ?m) (neg (at B ?l))
             (forall (?z - physob)
                (when (in ?z)
                      (and (at ?z ?m)
                           (neg (at ?z ?l)))))) )
(:action put-in
   :paras (?x - physob ?l  loc)
   :pre (and (neq ?x B)(at ?x ?l)(at B ?l))
             (forall (?z - physob)
                     (not (in ?z))))
   :eff (in ?x) )
(:action take-out
   :paras (?x - physob)
   :pre (in ?x)
   :eff (neg (in ?x)) )
(:goal move-object
   :paras (?x - physob ?l ?m - loc)
   :goal-des (and (neq ?l ?m)
                  (neq ?x B)
                  (imply (neg (at ?x ?l))
                         (at ?x ?m))) )
(:goal keep-object-at
   :paras (?x - physob ?l - loc)
   :goal-des (and (neq ?x B)
                  (imply (at ?x ?l)
                         (not (in ?x)))) )
(:goal keep-object-in
   :paras (?x - physob)
   :goal-des (in ?x) )
```

Figure 1: The action and goal schemata of the extended briefcase domain

```
(:action mov-b-1
   :paras (?l ?m - loc)
   :pre (and (neq ?l ?m) (at B ?l))
   :eff (and (at B ?m) (neg (at B ?l))
             (when (in B)
                   (and (at B ?m)
                        (neg (at B ?l))))
             (when (in C)
                   (and (at C ?m)
                        (neg (at C ?l))))
             (when (in D)
                   (and (at D ?m)
                        (neg (at D ?l))))) )
```

In the second step, the conditional effects in `mov-b-1` are further eliminated. Assume that, at this time step, the following propositions are true: `(at B H)`, `(at C H)`, `(at D H)`, and `(in D)`. Those conditional effects in `mov-b-1`, whose antecedents are not satisfied at the time step, are removed. We therefore have action schema `mov-b-2`.

```
(:action mov-b-2
   :paras (?l ?m - loc)
   :pre (and (neq ?l ?m) (at B ?l))
   :eff (and (at B ?m) (neg (at B ?l))
             (when (in D)
                   (and (at D ?m)
                        (neg (at D ?l))))) )
```

The antecedent of the remaining conditional effect in `mov-b-2` is already satisfied at the time step and it is moved into the existing preconditions. We finally have action schema `mov-b-3` at the current time step. Action schema `mov-b-3` is equivalent to the original action schema `mov-b` at the current time step. It is `mov-b-3` that is actually used for the action schema, '*Moving the briefcase from one location to another*', at the time step.

```
(:action mov-b-3
   :paras (?l ?m - loc)
   :pre (and (neq ?l ?m) (at B ?l)
             (in D))
   :eff (and (at B ?m) (neg (at B ?l))
             (at D ?m) (neg (at D ?l))))
```

The universally quantified goal descriptions in a goal schema are treated in the same way as the universally quantified preconditions in an action schema.

4. Goal Graphs, Valid Plans and Consistent Goals

In this section, we first describe the structure of the Goal Graph. We then define what we mean when we say a set of observed actions forms a valid plan for achieving a goal given

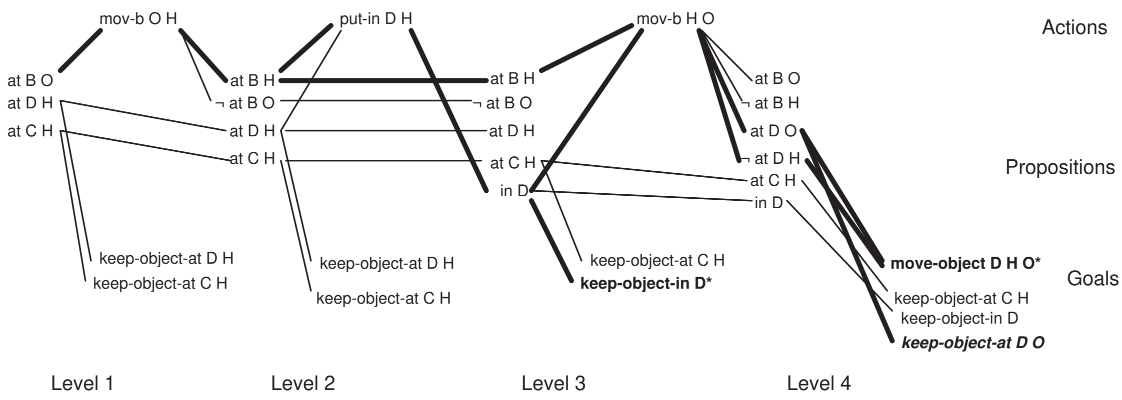

Figure 2: A Goal Graph for an example of the extended briefcase domain

the Initial Conditions. We finally define what we mean when we say a goal is consistent with a set of observed actions.

4.1 Goal Graphs

A Goal Graph represents the actions observed, the propositions true or explicitly known to be false, and the fully or partially achieved goals at consecutive time steps. A Goal Graph also explicitly represents connections between propositions, actions, and goals in the graph.

A Goal Graph is a directed, levelled graph. The levels alternate between proposition levels containing proposition nodes (each labelled with a proposition or negation of a proposition), representing the state of the world at consecutive time steps; goal levels containing goal nodes (each labelled with a goal), representing goals fully or partially achieved at consecutive time steps; and action levels containing action nodes (each labelled with an action), representing actions observed at consecutive time steps. The levels in a Goal Graph start with a proposition level at time step 1, consisting of one node for each proposition true in the Initial Conditions. They end with a goal level at the last time step, consisting of a node for each of the goals either fully or partially achieved by the actions observed so far. These levels are: propositions true at time step 1, goals achieved at time step 1, actions observed at time step 1; propositions true or explicitly known to be false at time step 2, goals achieved at time step 2, actions observed at time step 2; propositions true or explicitly known to be false at time step 3, goals achieved at time step 3, and so forth.

The goal nodes in goal-level i are connected by description edges to their goal descriptions in proposition-level i. The action nodes in action-level i are connected by precondition edges to their preconditions in proposition-level i, and by effect edges to their effects in proposition-level $i+1$. Those proposition nodes in proposition-level i are connected via persistence edges to the corresponding proposition nodes in proposition-level $i+1$, if their truth values have not been affected by the actions in action-level i. These persistence edges represent the effects of maintenance actions that simply bring forward proposition nodes in proposition-level i, not affected by the actions in action-level i, to proposition-level $i+1$.

In the example shown in Figure 2, three actions have been observed at three consecutive time steps: `(mov-b O H)`, `(put-in D H)`, and `(mov-b H O)`. The Initial Conditions consist of: (at B O), (at D H), and (at C H). Action and goal nodes are on the top and bottom parts of the graph respectively. The proposition nodes are in the middle part of the graph. The edges connecting proposition nodes and the action node in the same level are precondition edges. The edges connecting the action node in one level and propositions in a subsequent level are effect edges. The edges connecting proposition nodes and goal nodes in the same level are description edges. The edges connecting proposition nodes in one level to proposition nodes in a subsequent level are persistence edges. The goal nodes in bold represent consistent goals, among which the goal nodes in italics represent partially achieved goals, while the others represent fully achieved goals. The edges in bold show causal link paths. The goal nodes with an asterisk represent the recognised goals.

4.2 Valid Plans

We now define what we mean when we say a set of observed actions forms a valid plan for a goal, given the Initial Conditions.

Definition 1 (Causal Link) *Let a_i and a_j be two observed actions at time steps i and j respectively, where $i < j$. There exists a* **causal link** *between a_i and a_j, written as $a_i \rightarrow a_j$, if and only if one of the effects of a_i satisfies one of the preconditions of a_j.*

For instance, in the example shown in Figure 2, there exists a causal link between actions (mov-b O H) at time step 1 and (put-in D H) at time step 2, since one of the effects of the first action, (at B H), satisfied one of the preconditions of the second action.

A goal can be treated as an action with goal descriptions as its preconditions and an empty set of effects. Therefore, causal links can also be established from observed actions to goals.

For instance, in the example shown in Figure 2, there exists a causal link between action (mov-b H O) at time step 3 and goal (move-object D H O) at time step 4, since one of the effects of action, (at D O), satisfied one of the goal descriptions of the goal.

Now a valid plan for a goal can be defined on the basis of temporal ordering constraints and causal links over a set of observed actions. A valid plan P for a goal g, given the Initial Conditions, is represented as a 3-tuple, $< A, O, L >$, in which A is a set of observed actions, O is a set of temporal ordering constraints over A, and L is a set of causal links over A.

Definition 2 (Valid Plan) *Let g be a goal, and $P =< A, O, L >$, where A is a set of observed actions, O is a set of temporal ordering constraints, $\{a_i < a_j\}$, over A, and L is a set of causal links, $\{a_i \rightarrow a_j\}$, over A. Let I be the Initial Conditions. P is a* **valid plan** *for g, given I, if and only if*

1. *the actions in A can be executed in I in any order consistent with O;*
2. *the goal g is fully achieved after the actions in A are executed in I in any order consistent with O.*

For instance, in the example shown in Figure 2, given the Initial Conditions, $I = \{$(at B O), (at D H), (at C H)$\}$, $P = (\{a_1 =$ (mov-b O H), $a_2 =$ (put-in D H), $a_3 =$ (mov-b H O)$\}$, $\{a_1 < a_2,\ a_2 < a_3\}$, $\{a_1 \rightarrow a_2,\ a_1 \rightarrow a_3,\ a_2 \rightarrow a_3\})$ is a valid plan for goal $g =$ (move-object D H O).

4.3 Consistent Goals

We now define what we mean when we say a goal is consistent with a set of observed actions. A set of observed actions is represented by a 2-tuple, $< A, O >$, in which A is a set of observed actions and O is a set of temporal ordering constraints, $\{a_i < a_j\}$, over A.[4]

Definition 3 (Relevant Action) *Given a goal g and a set of observed actions, $< A, O >$, an action $a \in A$ is said to be* **relevant** *to g in the context of $< A, O >$, if and only if*

1. *there exists a causal link, $a \rightarrow g$; or*
2. *there exists a causal link, $a \rightarrow b$, where $b \in A$ is relevant to g and $a < b$ is consistent with O.*

Definition 4 (Consistent Goal) *A goal g is* **consistent** *with a set of observed actions, $< A, O >$, if and only if the strict majority of $a \in A$ are relevant to g in the context of $< A, O >$.*

Proposition 1 (Valid Plan for Consistent Goal) *Let $< A, O >$ be a set of observed actions, I be the Initial Conditions of $< A, O >$, g be a goal consistent with $< A, O >$. There exists a set of causal links, $L = \{a_i \rightarrow a_j\}$, over A and given I, $P =< A, O, L >$ is a valid plan for either g when g is fully achieved in the time step after $< A, O >$ has been observed or the achieved part of g when g is partially achieved in the time step after $< A, O >$ has been observed.*

Proof. When g is fully achieved in the time step after a set of actions has been observed, it directly follows Definitions 3 and 4 that there exists a set of causal links, $L = \{a_i \rightarrow a_j\}$, over A. It then follows Definition 2 that given I, $P =< A, O, L >$ is a valid plan for g.

When g is partially achieved in the time step after a set of actions has been observed, let $g\prime$ be the achieved part of g. So $g\prime$ is fully achieved in the time step after the set of actions has been observed, and it directly follows Definitions 2, 3 and 4 that there exists a set of causal links, $L = \{a_i \rightarrow a_j\}$, over A and given I, $P =< A, O, L >$ is a valid plan for $g\prime$. □

For instance, in the example shown in Figure 2, we have $< A, O > = < \{a_1 =$ `(mov-b O H)`, $a_2 =$ `(put-in D H)`, $a_3 =$ `(mov-b H O)`$\}, \{a_1 < a_2, a_2 < a_3\} >$, and $g =$ `(move-object D H O)` is a fully achieved goal in the time step after $< A, O >$ has been observed. According to Definition 3 and 4, g is consistent with $< A, O >$ because there exist causal links, $a_3 \rightarrow g$ between a_3 and g, $a_2 \rightarrow a_3$ between a_2 and a_3, $a_1 \rightarrow a_3$ between a_1 and a_3, and $a_1 \rightarrow a_2$ between a_1 and a_2. Let I be the Initial Conditions of $< A, O >$, $L = \{a_1 \rightarrow a_2, a_1 \rightarrow a_3, a_2 \rightarrow a_3\}$, according to Proposition 1, $P =< A, O, L >$ is a valid plan for g. Furthermore, causal link, $a_3 \rightarrow g$, explains the purpose of a_3.

In summary, according to Definition 4 and Proposition 1, when we say a goal is consistent with a set of observed actions, we mean that the strict majority of the observed actions are relevant to the goal and the set of observed actions forms a valid plan for the goal or the achieved part of it.

4. We assume that actions are observed at consecutive time steps but more than one action can be observed at a time step.

5. Goal Recognition Algorithms

We now describe our goal recognition algorithms. Our goal recognition algorithms run in a two-stage cycle at each time step. In the first stage, the GoalGraphConstructor algorithm takes the actions observed in the time step and tries to extend the Goal Graph. In the second stage, the GoalGraphAnalyser algorithm analyses the constructed Goal Graph to recognise those fully or partially achieved goals, that are consistent with the actions observed so far, and the valid plans for these goals or part of them. This two-stage cycle continues until no action is observed at the next time step.

5.1 Constructing a Goal Graph

We use a 4-tuple $< P, A_O, G_R, E >$ to represent a Goal Graph, where P is a set of proposition nodes, A_O is a set of action nodes, G_R is a set of goal nodes, and E is a set of edges. A proposition node is represented by $prop(p, i)$, where p is a positive or negative ground literal, i is a time step. An action node is represented by $action(a, i)$, where a is an observed action and i is a time step. A goal node is represented by $goal(g, i)$, where g is a goal and i is a time step. A precondition edge is represented by *precondition-edge*$(prop(p, i), action(a, i))$, an effect edge is represented by *effect-edge*$(action(a, i), prop(p, i+1))$, a description edge is represented by *description-edge*$(prop(p, i), goal(g, i))$, and a persistence edge is represented by *persistence-edge*$(prop(p, i-1), prop(p, i))$.

The GoalGraphConstructor algorithm consists of two algorithms: the goal expansion algorithm and the action expansion algorithm. The GoalGraphConstructor algorithm starts with a Goal Graph, $< P, \{\}, \{\}, \{\} >$, that consists of only proposition-level 1 with nodes representing the Initial Conditions.

Given a Goal Graph ending with proposition-level i, the goal expansion algorithm first extends the Goal Graph to goal-level i, with nodes representing goals fully or partially achieved at time step i. The algorithm goes through every possible ground instance of goal schemata. For every goal instance, it first gets a set of goal descriptions. It then eliminates all the universally quantified goal descriptions by dynamic expansion to get an equivalent set of goal descriptions. A goal node is added onto goal-level i to represent an achieved goal, if at least one of its goal descriptions has been satisfied at proposition-level i. It can then be decided whether a goal is fully or partially achieved, based on whether all or some of its goal descriptions have been satisfied respectively at proposition-level i. Meanwhile, if a node in proposition-level i satisfies a goal description, a description edge connecting the proposition node and the goal node is added onto the Goal Graph. Figure 3 shows the goal expansion algorithm. The algorithm takes a Goal Graph $< P, A_O, G_R, E >$ ending with proposition-level i, time step i, and a set of goal schemata G as input. It returns an updated Goal Graph ending with goal-level i after the goal expansion.

When actions are observed at time step i, the action expansion algorithm then extends the Goal Graph ending with goal-level i, to action-level i, with nodes representing the observed actions. At the same time, the algorithm also extends the Goal Graph to proposition-level $i+1$, with nodes representing propositions true or explicitly known to be false after the actions have been observed.

For every action observed at time step i, the algorithm first instantiates an action schema with the observed action to get a precondition set and an effect set. It then eliminates

Goal-Expansion($< P, A_O, G_R, E >$, i, G)
1. For every $G_k \in G$
   For every instance g of G_k
   a. Get a set of goal descriptions S_g.
   b. Get the equivalent set of S_g, $S_{g\prime}$.
   c. For every $p_g \in S_{g\prime}$, where $p_g = not(p_{g\prime})$,
      If $prop(neg(p_{g\prime}), i) \in P$, then
        Add *description-edge*($prop(neg(p_{g\prime}), i)$, $goal(g, i)$) to E.
   d. For every $p_g \in S_{g\prime}$, where $p_g \neq not(p_{g\prime})$,
      If $prop(p_g, i) \in P$, then
        Add *description-edge*($prop(p_g, i)$, $goal(g, i)$) to E.
   e. If one of the goal descriptions of g is satisfied, then
      Add $goal(g, i)$ to G_R.
2. Return with $< P, A_O, G_R, E >$.

Figure 3: The goal expansion algorithm

all the universally quantified preconditions and effects, as well as conditional effects, by dynamic expansion to get equivalent precondition and effect sets. Meanwhile, if a node in proposition-level i satisfies a precondition of the action, a precondition edge, connecting the proposition node and the action node, is added onto the Goal Graph. For every effect of the action, the action expansion algorithm simply adds a proposition node representing the effect to proposition-level $i + 1$. The effect edge from the action node to the proposition node is also added onto the Goal Graph.

After the above expansion, every proposition node at proposition-level i is brought forward to proposition-level $i + 1$ by a maintenance action, if its truth value has not been changed by an action observed at time step i (and it has not been added onto the Goal Graph by an action observed at time step i).[5] Persistence edges, connecting the corresponding proposition nodes at the two proposition levels, are added onto the Goal Graph.

Figure 4 shows the action expansion algorithm. The algorithm takes a Goal Graph $< P, A_O, G_R, E >$ ending with goal-level i, the set of actions observed at time step i, A_i, time step i, and a set of action schemata A as input. It returns an updated Goal Graph ending with proposition-level $i + 1$ after the action expansion. The expansion of the Goal Graph from proposition-level i to proposition-level $i + 1$ simulates the effects of executing the actions observed at time step i.

If otherwise there is no action observed at time step i, the GoalGraphConstructor algorithm finishes with nodes in goal-level i, representing all the possible goals either fully or partially achieved after all these actions have been observed.

5. Our goal recognition algorithms allow redundant actions.

Action-Expansion($< P, A_O, G_R, E >$, A_i, i, A)
1. For every $a_i \in A_i$
   a. Add $action(a_i, i)$ to A_O.
   b. Instantiate an action schema in A with a_i to get a precondition set S_P, and an effect set S_E.
   c. Get the equivalent sets of S_P and S_E, $S_{P'}$ and $S_{E'}$.
   d. For every $p_p \in S_{P'}$, where $p_p = not(p_{p'})$,
      If $prop(neg(p_{p'}, i) \in P$, then
      Add *precondition-edge*$(prop(neg(p_{p'}, i), action(a_i, i))$ to E.
   e. For every $p_p \in S_{P'}$, where $p_p \neq not(p_{p'})$,
      If $prop(p_p, i) \in P$, then
      Add *precondition-edge*$(prop(p_p, i), action(a_i, i))$ to E.
   f. For every $p_e \in S_E$
      i. Add $prop(p_e, i + 1)$ to P.
      ii. Add *effect-edge*$(action(a_i, prop(p_e, i + 1))$ to E.
2. For every $prop(p, i) \in P$
   If $prop(\neg p, i + 1) \notin P$, then
   If $prop(p, i + 1) \notin P$, then Add $prop(p, i + 1)$ to P;
   Add *persistence-edge*$(prop(p, i), prop(p, i + 1))$ to E.
3. Return with $< P, A_O, G_R, E >$.

Figure 4: The action expansion algorithm

Theorem 1 (Polynomial Size and Time) *Consider a goal recognition problem with s observed actions in t time steps, a finite number of objects at each time step, p propositions in the Initial Conditions, and m goal schemata each having a constant number of parameters. Let l_1 be the largest number of effects of any action schema, and l_2 be the largest number of goal descriptions of any goal schema. Let n be the largest number of objects at all time steps. Then, the size of the Goal Graph of $t + 1$ levels created by the GoalGraphConstructor algorithm, and the time needed to create the graph, are polynomial in n, m, p, l_1, l_2, and s.*

Proof. The maximum number of nodes in any proposition level is $O(p + l_1 s)$. Let k be the largest number of parameters in any goal schema. Since any goal schema can be instantiated in at most n^k distinct ways, the maximum numbers of nodes and edges in any goal level are $O(mn^k)$ and $O(l_2 mn^k)$ respectively. It is obvious that the time needed to create both nodes and edges in any level is polynomial in the number of nodes and edges in the level. □

Theorem 2 *The GoalGraphConstructor algorithm is sound: Any goal it adds to the Goal Graph at time step i is one either fully or partially achieved at time step i in the state of the world. The algorithm is complete: If a goal has been either fully or partially achieved by the observed actions up to time step $i - 1$, then the algorithm will add it to the Goal Graph at time step i, under the assumption that all the possible goals are restricted to the categories of goal schemata.*

Proof (soundness). Proposition-level 1 of the Goal Graph consists of only the Initial Conditions, representing the state of the world at time step 1 before any action has been observed. The Goal Graph is extended from proposition-level $i-1$ to proposition-level i, by adding only the effects of the actions observed at time step $i-1$, and bringing forward all the other proposition nodes that have not been affected by these actions from proposition-level $i-1$ to proposition-level i. Therefore, proposition-level i of the Goal Graph represents the state of the world at time step i, that has been changed from the Initial Conditions after the actions have been observed at time steps 1, ..., i −1.

A goal added to the Goal Graph at time step i by the algorithm is a fully or partially achieved goal in proposition-level i of the Goal Graph. Therefore, it is a goal that is fully or partially achieved in the state of the world at time step i.

Proof (completeness). Suppose a goal has been either fully or partially achieved by the actions observed at time steps 1, ..., i −1. This goal is then either fully or partially achieved in proposition-level i of the Goal Graph. Since goal-level i of the Goal Graph consists of all the possible instances of the goal schemata, that are fully or partially achieved in proposition-level i of the Goal Graph, and the goal is an instance of a goal schema, it is one of the fully or partially achieved goal instances in proposition-level i. The algorithm will therefore add the goal to goal-level i in the Goal Graph. □

5.2 Recognising Consistent Goals and Valid Plans

The GoalGraphAnalyser algorithm analyses the constructed Goal Graph to recognise consistent goals and valid plans. We assume that the strict majority of the observed actions are relevant to the goal intended by the agent in the context of the agent's actions. Therefore, the goal intended by the agent is consistent with the set of observed actions, and a goal may be the intended goal if it is consistent with the set of observed actions. In order to decide whether a goal is consistent with a set of observed actions, that is, whether it is relevant to the strict majority of the observed actions, we need to recognise causal links between either two observed actions or an observed action and the goal. We now define two particular types of paths, we call causal link paths, in the constructed Goal Graph. We prove in Theorems 3 and 4 that causal links can be recognised by identifying causal link paths.

Definition 5 *Given a Goal Graph, let a_i be an action observed at time step i and g_j be a goal fully or partially achieved in time step j, where $i < j$. A path that connects a_i to g_j via an effect edge, zero or more persistence edges, and a description edge, is called a causal link path between a_i and g_j.*

Theorem 3 *Given a Goal Graph, there exists a causal link, $a_i \rightarrow g_j$, between an action a_i at time step i and a goal g_j at time step j, where $i < j$, if a_i is connected to g_j via a causal link path.*

Proof. According to Definition 5, a causal link path between a_i and g_j consists of an effect edge, zero or more persistence edges, and a description edge. The effect edge on the path connects a_i to a proposition node in proposition-level $i+1$, representing one of the effects of a_i. When $j = i+1$, there is no persistence-edge on the path and this proposition node is connected to g_j by a description edge. When $j > i + 1$, this proposition node is brought forward to proposition-level j via $j-i-1$ persistence-edges by $j-i-1$ maintenance actions, and the brought-forward proposition node in proposition-level j is connected to g_j by a description edge. In either case, one of the effects of a_i satisfied one of the goal descriptions of g_j. Since a goal can be treated as an action with the goal descriptions as the preconditions and an empty set of effects, according to Definition 1, there exists a causal link between a_i and g_j. □

Definition 6 *Given a Goal Graph, let a_i and a_j be two actions observed at time steps i and j respectively, where $i < j$. A path that connects a_i to a_j via an effect edge, zero or more persistence edges, and a precondition-edge, is called a causal link path between a_i and a_j.*

Theorem 4 *Given a Goal Graph, there exist a causal link, $a_i \rightarrow a_j$, between an action a_i at time step i and an action a_j at time step j, where $i < j$, if a_i is connected to a_j via a causal link path.*

The proof of Theorem 4 is similar to Theorem 3. The details of the proof are omitted.

Given a constructed Goal Graph $< P, A_O, G_R, E >$ of t levels, the GoalGraphAnalyser algorithm shown in Figure 5 recognises every consistent goal from the goals in goal-level t, by deciding whether the strict majority of the observed actions are relevant to it. This is done by first finding those relevant actions from the observed actions, that are connected to the goal by causal link paths. For each of the already-known relevant actions, the algorithm tries to find more relevant actions from the observed actions, that are connected to it by causal link paths. This continues until no more relevant action is found. The consistent goal recognised and the valid plan for the goal or part of it are represented by a 3-tuple, $< g_t, < A_O, O, L_a >, L_g >$, where g_t is the goal, L_a is a set of causal links over the observed actions, and L_g is a set of causal links between some of the observed actions and the goal. $< A_O, O, L_a >$ represents a valid plan for g_t or part of it, and L_g further explains the purposes of some of the observed actions.

Proposition 2 *The GoalGraphAnalyser algorithm is sound: Any goal g it recognises at time step t is consistent with the observed actions so far, and the plan it organises for g or part of g is valid.*

Proof. The GoalGraphAnalyser algorithm recognises a goal g at time step t, when the strict majority of the observed actions are connected to g, either directly by a causal link path or indirectly by a chain of causal link paths. When an observed action is connected to g directly by a causal link path, according to Theorem 3 and Definition 3, there exists a causal link between the observed action and g, and the observed action is relevant to g. When an observed action is connected to g indirectly by a chain of causal link paths, according to Theorem 3, Theorem 4, and Definition 3, there is a chain of causal links between the observed action and g, and the observed action is relevant to g. Since the strict majority of the observed actions are relevant to g, according to Definition 4, g is consistent with the set of observed actions. Furthermore, according to Proposition 1, the plan the GoalGraphAnalyser algorithm organises for g or part of g, $< A_O, O, L_a >$, is a valid plan.□

GoalGraphAnalyser($< P, A_O, G_R, E >, t$)
1. For every $g_t \in G_R$ in goal-level t
   a. $A_{O'} \leftarrow \{\}$, $A \leftarrow \{\}$, $L_g \leftarrow \{\}$, $L_a \leftarrow \{\}$.
   b. For every $a_i \in A_O$ connected to g_t by a causal link path
      Add $a_i \rightarrow g_t$ to L_g; and
      Add a_i to $A_{O'}$; and
      Add a_i to A.
   c. If $A = \{\}$ and for most of $a_i \in A_O$, $a_i \in A_{O'}$, then
      Get all the ordering constraints, O, over A_O; and
      Add $< g_t, < A_O, O, L_a >, L_g >$ to GoalPlan.
   d. If $A \neq \{\}$, then
      Remove an action a_j from A; and
      For every $a_i \in A_O$ connected to a_j by a causal link path
        Add $a_i \rightarrow a_j$ to L_a; and
        If $a_i \notin A_{O'}$, then Add a_i to $A_{O'}$, a_i to A; and
      Go to 1c.
2. Return with GoalPlan.

Figure 5: The GoalGraphAnalyser algorithm

In the example shown in Figure 2, the goal nodes in bold represent three consistent goals, among which the goal node in italics represents a partially achieved goal, while the other two represent two fully achieved goals. The edges in bold show causal link paths.

Theorem 5 (Polynomial Space and Time) *Consider a t-level Goal Graph. Let l_1 be the number of fully or partially achieved goals at time step t, m_1 be the largest number of goal descriptions in any of these goals, l_2 be the number of the observed actions, and m_2 be the largest number of preconditions in any of these actions. The space size of possible causal link paths, that connect the goals to the observed actions and that connect the observed actions to other observed actions, and the time needed to recognise all the consistent goals, are polynomial in l_1, l_2, m_1, and m_2.*

Proof. Persistence edges do not branch in a Goal Graph. For each of the goals in goal-level t, the maximum number of paths searched for those observed actions, that are connected to the goal by causal link paths and hence relevant to it, is $O(m_1)$. For each of the relevant actions to the goal, the maximum number of paths searched for those observed actions, that are connected to the relevant action by causal link paths and hence also relevant to the goal, is $O(m_2)$. There are at maximum only l_1 goals in goal-level t and l_2 relevant actions to any of these goals. So the space size of possible causal link paths is $O(l_1(m_1 + l_2 m_2))$. The time needed to recognise all the consistent goals is polynomial in the space size. □

5.3 Goal Redundancy

The GoalGraphAnalyser algorithm recognises fully or partially achieved goals at a time step, that are consistent with the actions observed so far. Among these consistent goals, fully achieved goals better explain the actions observed so far. For instance, in the example shown in Figure 2, two consistent goals are recognised by the GoalGraphAnalyser algorithm at time step 4: `(move-object D H O)` is fully achieved and `(keep-object-at D O)` is partially achieved. Between these two consistent goals, the fully achieved goal better explains the observed actions so far. If, for instance, another action `(take-out D)` is observed at next time step, `(keep-object-at D O)` becomes fully achieved and remains consistent with the observed actions. So at that time step, it best explains the observed actions. A partially achieved goal, that is consistent with the observed actions so far, can remain consistent when more actions are observed in the future and becomes fully achieved. So choosing the fully achieved goal and making the partially achieved goal redundant does not rule out the possibility of the partially achieved goal remaining consistent and becoming fully achieved in the future. Based on this principle, we can make a partially achieved consistent goal at a time step redundant, if its satisfied goal descriptions are implied by the satisfied goal descriptions of another fully or partially achieved consistent goal.

Definition 7 *A partially achieved consistent goal at a time step is redundant, if the set of its satisfied goal descriptions is either a subset of the goal descriptions of a fully achieved consistent goal or a proper subset of the satisfied goal descriptions of another partially achieved consistent goal at the same time step.*

For instance, at time step 4 the set of satisfied goal descriptions of `(keep-object-at D O)` is a subset of the goal descriptions of `(move-object D H O)`. The partial achievement of `(keep-object-at D O)` has been implied by the full achievement of `(move-object D H O)`. So `(keep-object-at D O)` is made redundant by `(move-object D H O)` at time step 4.

A fully achieved consistent goal at a time step, however, can be made redundant only if its goal descriptions are implied by the goal descriptions of another fully achieved consistent goal at the same time step.

Definition 8 *A fully achieved consistent goal at a time step is redundant, if the set of its goal descriptions is a subset of the goal descriptions of another fully achieved consistent goal at the same time step.*

5.4 The Most Consistent Goals

After all the redundant goals have been removed from the set of consistent goals at a time step, there might still be more than one consistent goal in the set. If this is the case, the numbers of observed actions that are relevant to these remaining consistent goals will be compared. Those remaining goals that have the maximum number of relevant actions will be chosen as the most consistent goals at the time step.

Definition 9 *Given a set of consistent goals at a time step, a consistent goal in the set is the most consistent goal in the set, if it has the maximum number of relevant actions among all the consistent goals in the set.*

For instance, in the example shown in Figure 2, if another action `(take-out D)` is observed at time step 4, both `(move-object D H O)` and `(keep-object-at D O)` are consistent goals at time step 5, and neither of them is redundant. `(keep-object-at D O)` is relevant to all the five observed actions, while `(move-object D H O)` is relevant to only four of the observed actions. According to Definition 9, `(keep-object-at D O)` is the most consistent goal at time step 5.

In the example shown in Figure 2, the goal nodes with an asterisk represent the consistent goals eventually remaining after the two processes of removing redundant goals and selecting the most consistent goals.

6. Experimental Results

We implemented our goal recognition algorithms in Prolog and tested them in terms of accuracy, efficiency, and scalability on a desktop with a Pentium III processor at 600 MHz. We tested our algorithms on a set of data in the UNIX domain collected at the University of Washington. To collect the data, the subjects were given goals described in English first and they then tried to achieve each goal by executing UNIX commands. The commands issued to UNIX by the subject and the responses from UNIX to these commands were recorded in the data set. Some of the commands issued by the subject were not valid and could not be executed in UNIX. So the responses from UNIX to these invalid commands were actually error messages. For each of the goals the subjects tried to achieve, they indicated success or failure with regard to the achievement of the goal.

There are 14 goals in the UNIX data set, and each of these goals was tried by 5 subjects on average. As shown in Table 1, these goals can be classified into four types. The first type of goals are those of locating a file that has some of the properties, such as extension, size, contents, ownership, date, word count, and compression. The second type of goals are those of locating a machine that has some of the properties, such as load and logged-in users. The third type of goals are those of locating a printer that has some of the properties, such as print jobs and out of paper. The fourth type of goals are those of compressing all or large files in a directory. For the fourth type of goals, universally quantified goal descriptions are needed in the corresponding goal schemata. In addition, there are also two compound goals, G_3 and G_9, that are the conjunctions of two goals of the second type.

To test our algorithms, the sequences of UNIX commands, recorded in the data set, were taken as the observed actions at consecutive time steps. We took only the valid commands, that were successfully executed in UNIX, and filtered out the invalid commands, that UNIX failed to execute and responded only in the error messages. We created 35 action schemata for a set of commonly used UNIX commands, including those executed by the subjects. We also created 249 goal schemata, including 129 file-search goal schemata (for goals of locating a file that has some properties), 15 non-file-search goal schemata (for goals of locating a machine or a printer that has some properties and goals of compressing all or large files in a directory) and 105 goal schemata of paired non-file-search goals. The 14 goals in the UNIX data set are the instances of some of these goal schemata. We first tested our algorithms with respect to accuracy and efficiency, that is, the number of goals remaining after a sequence of observed actions has been processed, and the average time

G_1	Find a file named 'core'.
G_2	Find a file that contains 'motivating' and whose name ends in '.tex'.
G_3	Find a machine that has low (< 1.0) load; and determine if Oren Etzioni is logged into the machine named chum.
G_4	Compress all large (> 10,000 bytes) files in the Testgrounds subdirectory tree.
G_5	Compress all files in the directory named 'backups' [Don't use *].
G_6	Find a large file (> 100,000 bytes) that hasn't been changed for over a month.
G_7	Find a file that contains less than 20 words.
G_8	Find a laser printer in Sieg Hall that has an active print job.
G_9	Find a Sun on the fourth floor that has low (< 1.0) load; and determine if Dan Weld is active on the machine named chum.
G_{10}	Find a printer that is out of paper.
G_{11}	Find a file named oldpaper in ~neal/Testgrounds subdirectory.
G_{12}	Find a file of length 4 in ~neal/Testgrounds subdirectory.
G_{13}	See if Dan Weld is logged into chum.
G_{14}	Find a machine that Dan Weld is logged into.

Table 1: The 14 goals in the UNIX data set collected at the University of Washington

taken to construct a Goal Graph, analyse the constructed Goal Graph, and run through a cycle of Goal Graph construction and analysis, when an action was observed at a time step.

Table 2 gives a summary of the empirical results showing the accuracy of our algorithms. The first column shows the goals the subject tried to achieve. The achieved goals were the goals fully or partially achieved after the last observed action had been processed. The consistent goals were the fully or partially achieved goals that were consistent with the sequence of observed actions.[6] The remaining goals were the goals that remained after the redundant goals had been removed and the most consistent goals had been selected. The last column shows whether the given goal was among the remaining goals.

As shown in Table 2, our algorithms successfully recognised 13 out of 14 given goals. They failed to recognise only one given goal, G_{10}, simply because the sequence of commands executed by the subject actually failed to achieve the goal. In terms of the UNIX data set, goal recognition occurs when our algorithms return a single, consistent goal. This occurred on G_2, G_4, G_7, G_8, G_9, G_{13}, and G_{14}. On G_1, G_3, G_6, G_{11}, and G_{12}, more than one goal was recognised, including the goal given to the subject. On G_1, G_6, G_{11}, and G_{12}, our algorithms recognised that the subject tried to find one of the files with some properties in the directory but did not know which file it was. For instance, on G_1, the intended goal was to find a file named 'core'. The subject successfully found the file named 'core' in the directory, `other`, by executing a command, `ls`, to list all files in the directory. Since there were other files, `greenmouse`, `paper.tex`, and `action.ps.Z`, in the same directory,

6. In our experiments on the UNIX data set, a goal, to which more than two third of the observed actions were relevant, was recognised as a consistent goal. The threshold on the number of relevant actions is dependent on the application domain though the strict majority of the actions must be relevant. If the threshold is too high, our algorithms might fail to recognise the intended goal. On the other hand, if it is too low, the set of recognised goals might be too large to provide much value because of great ambiguity on the intended goal.

goal	achieved goals	consistent goals	remaining goals	given goal recognised
G_1	15	15	4	$\surd$
G_2	26	6	1	$\surd$
G_3	14	4	2	$\surd$
G_4	56	18	1	$\surd$
G_5	46	33	6	$\surd$
G_6	107	47	4	$\surd$
G_7	85	4	1	$\surd$
G_8	6	4	1	$\surd$
G_9	22	5	1	$\surd$
G_{10}	9	0	0	$\times$
G_{11}	12	12	2	$\surd$
G_{12}	60	44	6	$\surd$
G_{13}	1	1	1	$\surd$
G_{14}	8	2	1	$\surd$

Table 2: Empirical results of the UNIX domain showing the accuracy of our algorithms

and all these files were also listed by the same command, our algorithms recognised four goals, finding a file named ‘core’, finding a file named ‘greenmouse’, finding a file named ‘paper’ with extension ‘tex’, and finding a file named ‘action’ with extension ‘ps’ which is also compressed. On G_3, our algorithms recognised that the subject tried to find one of the users on a machine but did not know who he was. This was as good as a human observer could do because you simply could not tell from the observed actions which file or user the subject was trying to find. These recognised goals can be generalised into a single, consistent goal ‘finding a file in the directory’ or ‘finding a user using a machine’, where variables are allowed in the recognised goals.

On G_5, our algorithms recognised 6 consistent goals. Among these goals, five goals were to find each of the five files with the same properties of compression and extension in the directory named ‘backups’. Another goal was to gunzip all files in the directory named ‘backups’. A human observer could probably do better by recognising the goal of gunzipping all files in the directory named ‘backups’, because it accounted better for the gunzip command that had been observed. It was very unlikely that the subject gunzipped all files in the directory in order to find a file with gunzip compression.

Among the goals we tested, G_1, G_2, G_3, and G_4 were originally tested by Lesh and Etzioni (1995). Our empirical results show a significant improvement on the accuracy of the goal recogniser implemented by them in terms of the remaining goals. Our algorithms have 4, 1, 2, and 1 remaining goals on G_1, G_2, G_3, and G_4 respectively, while their goal recogniser has 155, 37, 1, and 15 remaining goals on G_1, G_2, G_3, and G_4 respectively, after the last observed action has been processed. Furthermore, the 4 and 2 remaining goals that our algorithms have on G_1 and G_3 can be generalised into two single goals. These results show that our algorithms perform extremely well with regard to accuracy.

goal	length of observation	construction time	analysis time	time per cycle
G_1	2.25	0.535	0.013	0.547
G_2	16	0.382	0.102	0.484
G_3	3.0	0.021	0.009	0.030
G_4	20.5	1.036	3.609	4.645
G_5	9	0.426	0.565	0.991
G_6	8.78	12.185	4.143	16.329
G_7	9.11	16.473	8.581	25.054
G_8	3.5	0.014	0.002	0.017
G_9	12	0.034	0.050	0.084
G_{10}	15	0.018	0.007	0.025
G_{11}	7	0.051	0.020	0.071
G_{12}	17	0.821	0.774	1.595
G_{13}	1	0.010	0.001	0.011
G_{14}	2	0.013	0.004	0.017

Table 3: Empirical results of the UNIX domain showing the efficiency of our algorithms

It is worth noting that Lesh and Etzioni's algorithm converges before the last observed action has been processed. So their algorithm works towards goal prediction, while our algorithms emphasise the explanation of the observed actions, by recognising fully or partially achieved goals that are consistent with these actions. Their algorithm can quickly prune out inconsistent goals to get a converged set of hypothesised goals, though the number of hypothesised goals in the set can sometimes be large. The next step of their work might be to assign probabilities to these hypothesised goals to differentiate a single goal from the others, when there exists only one intended goal. On the other hand, it is less desirable for the goals recognised by our algorithms to be differentiated from each other as the accuracy of the algorithms is usually high. Our algorithms, however, cannot recognise a goal until it has been fully or partially achieved.

Table 3 gives a summary of the empirical results showing the efficiency of our algorithms. The length of observation was the average number of observed actions executed by the subjects to achieve the given goal. The construction time was the average time it took to construct the Goal Graph at a time step. The analysis time was the average time it took to analyse the constructed Goal Graph at a time step. The time per cycle was the average time it took to go through a two-stage cycle of Goal Graph construction and analysis, when an observed action was processed at a time step, including constructing the Goal Graph, recognising the consistent goals, removing redundant goals, and selecting the most consistent goals. As shown in Table 3, on average at a time step, it took 2.287 CPU seconds to construct the Goal Graph, 1.277 CPU seconds to analyse the Goal Graph, and 3.564 CPU seconds to process an observed action. Since our algorithms have been written in the less efficient Prolog and run on a desktop with a Pentium III processor at 600 MHz, more efficiency could be achieved. The construction time, analysis time, and time per cycle could

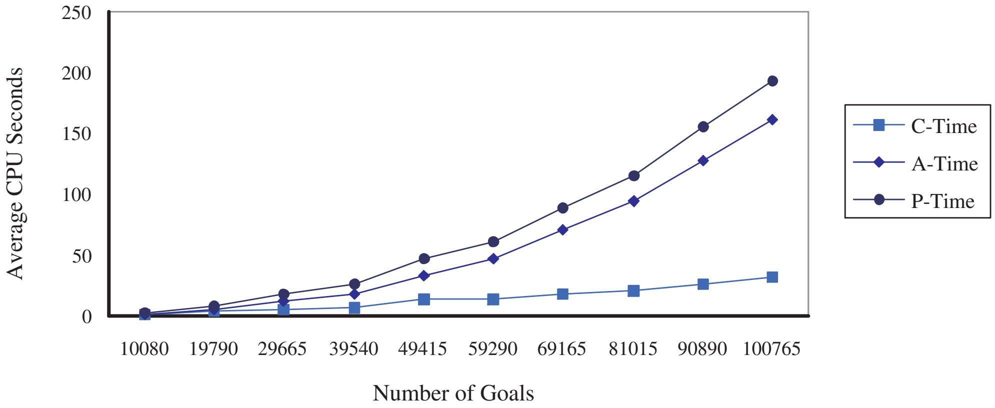

Figure 6: Empirical results of the UNIX domain showing the scalability of our algorithms

be reduced to well below a CPU second, if the algorithms are coded in a more efficient programming language and run on a faster machine.

Compared to the empirical results on the efficiency of the goal recogniser implemented by Lesh and Etzioni (1995), on average, it took 0.547, 0.484, 0.030, and 4.645 CPU seconds to process an observed action by our algorithms on G_1, G_2, G_3, and G_4 respectively, while 1.616, 1.643, 0.648, and 1.610 CPU seconds were taken by their goal recogniser to process an observed action on the same goals. The average time to process an observed action was roughly around 1.4 CPU seconds on both a desktop with a Pentium III processor at 600 MHz by our algorithms coded in Prolog and a SPARC 10 by their goal recogniser coded in Lisp. Given that two different programming languages and machines were used for the implementation of two different systems, this comparison is hardly meaningful. It is however apparent that the use of more efficient programming languages and machines can significantly speed up our algorithms.

We also tested the scalability of our goal recognition algorithms in the UNIX domain. We tested how the efficiency of our algorithms was affected by the number of possible goals, that was in turn affected by the number of objects in the universe of objects. We created a series of spaces of approximate 10^4, 2×10^4, 3×10^4, up to 10^5 possible goals respectively based on the data recorded on G_7 in the UNIX data set.[7] For doing this, We changed the files and directories in the file hierarchy, as well as the properties of the files, in the original data set for G_7, to increase or decrease the number of objects in the universe of objects, while keeping the files and directories related to the intended goal and the properties of the related files unchanged. In the sense that while part of the Initial Conditions on the intended goal remained the same, the rest of the Initial Conditions were changed to create

7. The number of goal schemata remains unchanged.

the appropriate number of candidate goals. The change in the file hierarchy reflected the change in the complexity of the file hierarchy.

The original sequences of commands recorded in the data set were used in the experiments, in conjunction with the different sets of the Initial Conditions, for the creation of different spaces of candidate goals. Figure 6 shows that the average CPU time taken at a time step to construct and analyse the Goal Graph (shown as C-Time and A-Time respectively in Figure 6), and to process the observed action as a whole (shown as P-Time in Figure 6) was approximately linear in the number of candidate goals.

7. Conclusion and Future Work

In this paper, we presented a new approach to goal recognition in which a graph structure called a Goal Graph is constructed and analysed for goal recognition. We described two algorithms for constructing and analysing a Goal Graph. Our algorithms recognise partially or fully achieved goals that are consistent with the observed actions, and reveal valid plans for the recognised goals or part of them. Our algorithms do not need a plan library. They allow redundant, extraneous, and partially ordered actions. They are sound, polynomial-time, and polynomial-space.

Our empirical experiments show that our algorithms recognise goals with great accuracy. They are computationally efficient. They can be scaled up and applied to domains where there are tens of thousands of goals and plans. Even though one of our goal recognition algorithms, the GoalGraphAnalyser algorithm, is not complete, they recognised all the intended goals in the UNIX data set that were successfully achieved by the subjects.

A limitation of our goal recognition algorithms is that sometimes more than one goal is recognised, though the number of recognised goals is usually very small. Our algorithms cannot tell which goal is the most probable one when there is only one intended goal. For instance, on G_5 in the UNIX data set, our algorithms recognised 6 goals. Even though the intended goal, gunzipping all files in the directory called `backups`, was among these goals, and it was probably the most likely one compared to the others, our algorithms could not differentiate it from the others. Sometimes our algorithms can recognise a unique goal that implies the intended goal but cannot make it more specific. For instance, on G_7 in the UNIX data set, our algorithms recognised a unique goal, finding the file, `index.tex`, with its word count and extension. Even though the achievement of this goal implied the achievement of the intended goal, finding a file that contains less than 20 words, our algorithms could not be more specific. These problems are due to the incomplete information we had from the observed actions. On G_7, the observed actions did not have any indication on the likelihood of the recognised goals being the intended goal. The subject actually achieved G_7 because he knew that the word count was less than 20 words by knowing the word count of the file, but no observed action was directly used to achieve this.

Several attempts (Carberry, 1990; Charniak & Goldman, 1993; Huber et al., 1994; Forbes et al., 1995; Pynadath & Wellman, 1995; Bauer, 1995; Albrecht et al., 1998) have been made to incorporate uncertainty representation and reasoning techniques into plan recognition for handling uncertain and incomplete information, so that a single plan can be distinguished from a set of candidate plans. However, these systems rely heavily on the availability of planning knowledge and to a certain extent the use of plan libraries. In future work, we intend to investigate the possibility of incorporating uncertainty representation and reasoning mechanisms into our Goal-Graph-based approach to goal recognition, so that a unique, specific goal can be recognised when there is only one intended goal, while all the good features of our Goal-Graph-based approach are kept.

In future work, we also intend to explore the extent to which our Goal Graph representation can be used for probabilistic goal recognition. In particular, we will consider problem settings in which the effects of actions are probabilistic and the objective of goal recognition is to recognise consistent goals with the highest probability of achievement.

Another area of future work is to recognise goals even before they have been partially or fully achieved by the actions observed so far. In this regard, our current goal recognition algorithms can be used to recognise goals that each sequence of actions in a very large data set can achieve. Then each sequence of actions in the data set and the recognised goals can be used to acquire a probabilistic model of goal prediction by a machine learning method. This probabilistic model takes the actions observed so far and predicts possible goals even when they are not partially or fully achieved.

Acknowledgments

This paper is a revised and extended version of a paper appeared in the Proceedings of AAAI-2000 (Hong, 2000). The author wishes to thank Neal Lesh for providing the UNIX data set, and anonymous referees for their insights and constructive criticisms, which have helped improve the paper significantly.

References

Albrecht, D. W., Zukerman, I., & Nicholson, A. E. (1998). Bayesian model for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, *8*(1-2), 5–47.

Allen, J. F. (1983). Recognizing intentions from natural language utterances. In Brady, M., & Berwick, B. (Eds.), *Computational Models of Discourse*, pp. 107–166. MIT Press, Cambridge, MA.

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

Anderson, C., Smith, D. E., & Weld, D. (1998). Conditional effects in Graphplan. In *Proceedings of the 4th International Conference on AI Planning Systems*, pp. 44–53.

Bauer, M. (1995). Dempster-Shafer approach to modeling agent preferences for plan recognition. *User Modeling and User-Adapted Interaction*, *5*(3-4), 317–348.

Bauer, M. (1998). Acquisition of abstract plan descriptions for plan recognition. In *Proceedings of AAAI-98*, pp. 936–941.

Bauer, M., & Paul, G. (1993). Logic-based plan recognition for intelligent help systems. In Bäckström, C., & Sandewall, E. (Eds.), *Current Trends in AI Planning*, pp. 60–73. IOS Press.

Blum, A. L., & Furst, M. L. (1997). Fast planning through Planning Graph analysis. *Artificial Intelligence*, *90*, 281–300.

Carberry, S. (1986). User models: the problem of disparity. In *Proceedings of the 11th International Conference on Computational Linguistics*, pp. 29–34.

Carberry, S. (1988). Modeling the user's plans and goals. *Computational Linguistics*, *14*(3), 23–37.

Carberry, S. (1990). Incorporating default inferences into plan recognition. In *Proceedings of AAAI-90*, pp. 471–478.

Carver, N. F., Lesser, V. R., & McCue, D. L. (1984). Focusing in plan recognition. In *Proceedings of AAAI-84*, pp. 42–48.

Charniak, E., & Goldman, R. P. (1993). A Bayesian model of plan recognition. *Artificial Intelligence*, *64*, 53–79.

Forbes, J., Huang, T., Kanazawa, K., & Russell, S. (1995). The BATmobile: Towards a Bayesian automated taxi. In *Proceedings of IJCAI-95*, pp. 1878–1885.

Gazen, B., & Knoblock, C. (1997). Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proceedings of the 4th European Conference on Planning*, pp. 221–233.

Goodman, B. A., & Litman, D. J. (1992). On the interaction between plan recognition and intelligent interfaces. *User Modeling and User-Adapted Interaction*, *2*, 83–115.

Grosz, B. J., & Sidner, C. L. (1990). Plans in discourse. In P. R. Cohen, J. M., & Pollack, M. E. (Eds.), *Intentions in Communication*, pp. 417–444. MIT Press, Cambridge, MA.

Hong, J. (2000). Goal Graph construction and analysis as a paradigm for plan recognition. In *Proceedings of AAAI-2000*, pp. 774–779.

Huber, M. J., & Durfee, E. H. (1995). Deciding when to commit to action during observation-based coordination. In *Proceedings of the First International Conference on Multi-Agent Systems*, pp. 163–170.

Huber, M. J., Durfee, E. H., & Wellman, M. P. (1994). The automated mapping of plans for plan recognition. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pp. 344–351.

Huff, K., & Lesser, V. (1988). A plan-based intelligent assistant that supports the software development process. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pp. 97–106.

Kautz, H. A. (1987). *A Formal Theory of Plan Recognition*. PhD Thesis, University of Rochester.

Koehler, J., Nebel, B., Hoffmann, J., & Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In *Proceedings of the 4th European Conference on Planning*, pp. 273–285.

Lesh, N., & Etzioni, O. (1995). A sound and fast goal recognizer. In *Proceedings of IJCAI-95*, pp. 1704–1710.

Lesh, N., & Etzioni, O. (1996). Scaling up goal recognition. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 178–189.

Litman, D. J., & Allen, J. F. (1987). A plan recognition model for sub-dialogues in conversation. *Cognitive Science*, *11*(2), 163–200.

Mooney, R. J. (1990). Learning plan schemata from observation: Explanation-based learning for plan recognition. *Cognitive Science*, *14*(4), 483–509.

Pednault, E. P. D. (1988). Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, *4*(4), 356–372.

Pednault, E. P. D. (1989). ADL: Exploring the middle ground between STRIPS and the Situation Calculus. In *Proceedings of the 1st International Conference on Knowledge Representation and Reasoning*, pp. 324–332.

Pollack, M. E. (1990). Plans as complex mental attitudes. In Cohen, P. R., Morgan, J., & Pollack, M. E. (Eds.), *Intentions in Communication*, pp. 77–101. MIT Press, Cambridge, MA.

Pynadath, D. V., & Wellman, M. P. (1995). Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pp. 472–481.

Schank, R., & Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding.* Erlbaum.

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

Wilensky, R. (1983). *Planning and Understanding.* Addison-Wesley Publishing Company, Reading, MA.

Wilensky, R., & et al. (1988). The Berkeley UNIX consultant project. *Computational Linguistics*, *14*(4), 35–84.