Partial Observability in Grammar Based Plan Recognition.

Christopher W. Geib
College of Computing and Informatics
Drexel University
3141 Chestnut St.,
Philadelphia, PA 19104, USA
cwgeib@drexel.edu

Abstract

Prior work on viewing plan recognition as parsing of grammars has assumed completely observable actions. This paper provides an algorithm to rewrite plan grammars to allow for recognizing partially observable actions. For the ELEXIR (Geib 2009) system, the impact of this rewriting on plan recognition runtime is shown to be limited to those plans that actually use the partially observable actions.

Introduction

Like human speech processing that requires both 1) the construction syntactic tokens from the raw signal and 2) parsing of these tokens into larger sentences, recognizing the complex plan structures being followed by intelligent agents requires two different processing stages. First we must convert raw, sensor reports of observed activities into a stream of discrete action tokens representing the agents movements. This first process, usually called *activity recognition*, is well addressed by a number of well known probabilistic reasoning methods like HMMs, CRFs, and POMDPs(Hoogs and Perera 2008; Liao, Fox, and Kautz 2005; Vail and Veloso 2008). However, in addition, a second process, usually called *plan recognition*, must assemble such activity tokens to produce complex, hierarchical plan structures that organize the completed parts of the plan that support higher level goals and even make predictions about future supporting activities.

While many of the algorithms for activity recognition have been shown to deal well with partially observable domains, the algorithms for plan recognition have not. For example, the Engine for LEXicalized Intent Recognition (ELEXIR) system (Geib 2009; Geib and Goldman 2011) views plan recognition as parsing of a stream of observed action tokens (executed by the agent) given a probabilistic plan grammar defined. Viewing plan recognition as probabilistic parsing has a number of advantages enumerated in prior work (Vilain 1990; Geib and Steedman 2007; Geib 2009; Geib and Goldman 2011). However, even given these advantages, the current work on ELEXIR has a significant limitation, it assumes that all of the actions in the domain can be observed with certainty. That is, none of the actions can be performed and not observed or can be incorrectly identified when they have been performed. This paper begins the process of removing this assumption.

ELEXIR's foundation in parsing formal grammars provides a simple method to address this problem. The grammar that captures the plan library can be rewritten to make the observation of the desired actions in the grammar optional, and the probability model of the grammar is changed to reflect the possibility of the missing observation of the action. No changes are required to the ELEXIR algorithm to deal with partially observable domains. Thus, our approach has no impact on the completeness or correctness of the plan recognition algorithm itself. However, the required rewriting of the plan grammar can impact system runtime.

The rest of this paper is structured as follows. First it will provide a brief overview of plan recognition as parsing in the form of ELEXIR, followed by a discussion of prior work. Then it will then discuss rewriting grammars to address partially observable actions and the impact of rewriting on system runtime.

Plan Recognition as Parsing

Vilain (Vilain 1991) suggested that plan recognition could be viewed as a kind of parsing like that performed to understand natural languages. Taking this suggestion seriously, the ELEXIR system (Geib 2009; Geib and Goldman 2011) represents plans using Combinatory Categorial Grammars (CCG) (Steedman 2000). In such a grammar, each *terminal* of the grammar, represents an observable action in the domain, and is associated with a number of *categories* that capture the structure of the possible plans in which it can be used. The set of all categories, C, is defined recursively:

Atomic categories : A finite set of basic action categories. $C = \{A, B, ...\}$. Note that for the rest of this paper all atomic categories will be denoted using capital letters.

Complex categories : $\forall Z \in C$, and non-empty set $\{W, X, ...\} \subset C$ then $Z\backslash\{W, X, ...\} \in C$ and $Z/\{W, X, ...\} \in C$.

Complex categories represent functions that take a set of *arguments* (the categories to the right of a slash, $\{W, X, ...\}$) and produce a *result* (the category to the left of the slash, Z). The direction of the slash indicates where the function should observe its arguments. Rightward slash arguments

must be observed after the action the category is assigned to, and leftward slash arguments must be observed before.

In addition to a *lexicon* that assigns categories to terminals of the grammar, ELEXIR uses three *combinators* (Curry 1977) defined over pairs of categories, to combine CCG categories into higher level plan structures:

$$\begin{array}{ll} \textit{rightward application:} & X/\alpha \cup \{Y\},\ Y \ \Rightarrow\ X/\alpha \\ \textit{leftward application:} & Y,\ X\backslash\alpha \cup \{Y\} \ \Rightarrow\ X\backslash\alpha \\ \textit{rightward composition:} & X/\alpha \cup \{Y\},\ Y/\beta \ \Rightarrow\ X/\alpha \cup \beta \end{array}$$

where X and Y are categories, and α and β are possibly empty sets of categories. To see how CCGs perform plan recognition, consider the following simple example. We can encode the fact that the sequence of actions: $[actA, actB, actC]$ is a valid plan to achieve goal G as:

CCG: 1

$$\begin{array}{l} actA :=\{\ A\ \} \\ actB :=\{\ ((G)/\{C\})\backslash\{A\}\ \} \\ actC :=\{\ C\ \} \end{array}$$

where A, C and G are atomic categories.

The optional parentheses highlights the left associative nature of complex categories. Note a lexicon can assign multiple categories to an action. However, within any single parse an action can only be assigned a single category. To see how a lexicon and combinators parse observations into high level plans, consider the derivation in Figure that parses the observation sequence: $[actA, actB, actC]$ using CCG: 1. Each observation is incrementally assigned one

$$\dfrac{\dfrac{\dfrac{\text{actA}}{A} \quad \dfrac{\text{actB}}{((G)/\{C\})\backslash\{A\}} \quad \dfrac{\text{actC}}{C}}{(G)/\{C\}}}{G}$$

Figure 1: Parsing Observations with CCG categories

of its categories from the lexicon. Combinators (rightward and leftward application in this case) then combine the categories.

We define an *explanation* of a sequence of observations as an ordered sequence of (possibly complex) categories that result from the ELEXIR parsing algorithm. As each distinct parse of the observations represents a distinct set of categories, or methods for combining the categories, they represent different "explanations" for the observed actions.

For any category we define the *root result* of the category. In the case of atomic categories it is the category itself. For complex categories it is the left-most, inner, atomic result category. We define a function *root* that returns the root result of any category. For example:

$$G = root(\ ((G)/\{C\})\backslash\{A\}\ )$$

Further, we will call an action an *anchor* for a category if the specified category is the root result of one of the categories in the action's definition. For example in CCG: 1, $actB$ is an anchor for category G and $actA$ is an anchor for A.

It will be helpful to be able to write expressions with category variables. A vertical bar will be used when we don't care about the direction of a category's slash operator. Bold, subscripted lowercase "v"s will be used as variables for single atomic categories. So $\mathbf{v_1}|\{B\}$ matches all complex categories that have only a single atomic category B argument on either side. (eg. $A/\{B\}$ or $C\backslash\{B\}$ but not $(G\backslash\{A\})\backslash\{B\}$).

Further, bold, subscripted, lowercase "v"s with a right arrow over them will be used as variables for possibly complete, complex category fragments (excluding their initial and final slashes). For example, $G|\overrightarrow{\mathbf{v_1}}|\{A\}$ will match any category with at least two arguments whose root result is G and whose first argument is A. This would include $(G\backslash\{B\})\backslash\{A\}$ as well as $((G/\{B\})/\{C\})/\{A\}$. The expression $\overrightarrow{\mathbf{v_1}}|\{B\}$ would match any complex category whose first argument is B.

The Probability Model

ELEXIR defines the probability of a goal as:

Definition 0.1

$$P(goal|obs) = \frac{\sum_{\{exp_i|goal\in exp_i\}} P(exp_i \wedge obs)}{\sum_{j=0}^{n} P(exp_j \wedge obs)}$$

Where $P(exp_i \wedge obs)$ is the probability of explanation exp_i and the observations. Thus, the conditional probability for any particular goal is the sum of the probability mass associated with those explanations that contain it divided by the probability mass for all the explanations of the observations.

ELEXIR defines the probability of an explanation, exp, with m categories, that explains n observations, $\sigma_1 ... \sigma_n$, as:

Definition 0.2

$$P(exp \wedge \{\sigma_1 ... \sigma_n\}) = \prod_{i=1}^{m} P(root(c_{exp,i})) \prod_{j=1}^{n} P(cinit_{exp,j}|\sigma_j)K$$

where $cinit_{exp,j}$ represents the initial category assigned in explanation exp to observation σ_j, and $root(c_{exp,i})$ represents the root result category of the ith category in the explanation and K is a normalizing constant.

Intuitively, the first term captures the prior probability of the agent having the root goals present in the explanation. That is, the root results of the categories in the explanation are the set of goals the agent is currently pursuing, and they will not be combined into some larger goal. The second term captures the probability that each observation is assigned its initial category given the set of possible categories the lexicon allows it to have. Note that this means that any ELEXIR lexicon must specify a probability distribution over the possible categories that an observation can be assigned.

Prior work (Geib and Goldman 2011) provides more detail the system and we direct the reader there for more detailed discussion of it

Related Work

Kautz' foundational, graph covering based work on plan recognition (Kautz 1991), in fact did not assume perfect observations, but instead fit the best vertex cover to the plan graph. However, it was unable to address a number of issues

that ELEXIR does address including multiple interleaved goals. Other similar early work using logic based reasoning algorithms (Carberry 1990; Litman 1986) did formalize the inference necessary for efficient plan recognition. However, addressing partially observable domains was not attempted.

As we have already noted, recent work on probabilistic activity recognition using HMMs and CRFs (Hoogs and Perera 2008; Liao, Fox, and Kautz 2005; Vail and Veloso 2008) is actually addressing a different problem than the plan recognition problem. These systems are looking for a single label for a sequence of observations. The object of plan recognition is to unite a number of such labels into a structured, higher-level plan. Similarly, Ramirez and Geffner (Ramirez and Geffner 2011) have proposed using POMDPs that could address partially observable actions for plan recognition. However, like the HMMs and CRFs, in their work they are attempting to infer a single high level goal that is the objective of an agent specified by a POMDP rather than the complex plan structures inferred in this work.

The work of Bui (Bui, Venkatesh, and West 2002) presents an interesting alternative to this work in that Hierarchical Hidden Markov Models should be able to address partially observable domains, and capture the hierarchy of plan structure. However, this prior work has difficulty dealing with multiple interleaved plans, an issue that is already addressed by ELEXIR. Further, the focus of this work is on extending grammar based methods of plan recognition.

ELEXIR follows the early work of Vilain (Vilain 1990) on plan recognition as parsing. However this early work does not actually present an algorithm or implemented system for plan recognition. Pynadath and Wellman (Pynadath and Wellman 2000) do formalize plan recognition based on probabilistic context free grammars (PCFGs). However they do not view plan recognition as a parsing problem. Instead, they use the structure of plans captured in a PCFG to build a fixed Dynamic Bayes Net (DBN), and use the resulting DBN to compute a distribution over the space of possible plans that could be under execution.

There are other recent pieces of work, that take a least commitment approach to plan recognition (Avrahami-Zilberbrand and Kaminka 2005) or attempt to bound the probabilities for gramatical methods (Kabanza et al. 2013) that could be modified to address partial observability. However, since this work is also based on recognizing a library of plans, to be complete, they would both be forced to do the same kind of plan rewriting we have described here. Thus, this work should really be seen as a opportunity for further application of these ideas.

Geib (Geib and Goldman 2005) has also attempted to address partial observability in a grammar based formalism, however, they addressed the problem by modifying the parsing algorithm to hypothesize the execution of all of actions that could possibly be executed at each time step. This is in addition to maintaining and extending the set of all of the hypothesis that are consistent with the action that is actually observed next in the series of observations. The additional computational cost of hypothesizing all of these unobserved actions makes this approach impractical for all but the simplest domains. Our approach makes no such changes to the parsing algorithm. The process of plan recognition remains driven solely by the the observed actions and therefore remains much more efficient.

Making an Action Partially Observable

The use of formal grammars in ELEXIR gives us a method for dealing with partial observability. Inspired by epsilon removal from grammars (Hopcroft and Ullman 1979), we can rewrite the grammar to make actions optional. For this discussion, we will define the *yield* of a particular ELEXIR plan lexicon, as the set of all sequences of observables that can be parsed to produce a single atomic category using the lexicon. Thus each element of the yield of a plan lexicon is a sequence of observables for a single plan within the lexicon.

Suppose we want to make *actB* partially observable. We must create a new grammar whose yields includes all the sequences in our initial grammar, and all of the sequences from the first grammar that contain *actB* with *actB* missing. Thus if $[actA, actB, actC]$ is in the yield of the initial grammar, our new grammar should also accept $[actA, actC]$. To do this we add categories to the lexicon for specific actions.

However extending the yield of the grammar is not sufficient. We must also change the probabilities associated with the grammar in order to compute the correct probabilities. Recall that each lexicon keeps a probability distribution over the set of categories that an action can be assigned during a parse. If we add categories to an action's definition then we must also change this probability distribution to reflect how likely it is that the new category is the one chosen for the action. We will discuss these two changes in order.

Extending the Grammar's Yield

Extending the yield of the grammar to account for partially observable actions is accomplished by adding new categories to specific entries in the lexicon. Suppose we want to make an action $actX := \{c_1^X, ...c_n^X\}$ partially observable. To build the new lexicon, we start from a complete copy of the original lexicon, and apply the following two rules.

1. For all actions in the lexicon $actY := \{c_1^Y, ...c_m^Y\}$ such that $actX \neq actY$. If $\exists\, A \in C$ and indicies i and j such that,
   (a) A is an atomic category, and
   (b) A is an argument of c_i^Y, and
   (c) $c_j^X = A|\overrightarrow{\mathbf{v_1}}$, then define

$$c_{new} = \begin{cases} \overrightarrow{\mathbf{v_2}}/\overrightarrow{\mathbf{v_1}}/\overrightarrow{\mathbf{v_3}} & \text{if } c_i^Y = \overrightarrow{\mathbf{v_2}}/\{A\}/\overrightarrow{\mathbf{v_3}}, \\ \overrightarrow{\mathbf{v_2}}/\overrightarrow{\mathbf{v_1}}\backslash\overrightarrow{\mathbf{v_3}} & \text{if } c_i^Y = \overrightarrow{\mathbf{v_2}}/\{A\}\backslash\overrightarrow{\mathbf{v_3}}, \\ \overrightarrow{\mathbf{v_2}}\backslash\overrightarrow{\mathbf{v_1}}/\overrightarrow{\mathbf{v_3}} & \text{if } c_i^Y = \overrightarrow{\mathbf{v_2}}\backslash\{A\}/\overrightarrow{\mathbf{v_3}}, \\ \overrightarrow{\mathbf{v_2}}\backslash\overrightarrow{\mathbf{v_1}}\backslash\overrightarrow{\mathbf{v_3}} & \text{if } c_i^Y = \overrightarrow{\mathbf{v_2}}\backslash\{A\}\backslash\overrightarrow{\mathbf{v_3}}. \end{cases}$$

   and add c_{new} to $actY$'s possible categories in the lexicon.
2. For every category $c_i^X = \overrightarrow{\mathbf{v_1}}|\{\mathbf{v_2}\}$, if $\exists\, G \in C$ such that,
   (a) $G = root(c_i^X)$, and
   (b) G does not occur as an argument to any other category in the lexicon, then

For each action $actY := \{c_1^Y, ...c_m^Y\}$ such that $actY \neq actX$,
For each c_j^Y such that $\mathbf{v_2} = root(c_j^Y)$

$$c_{new} = \begin{cases} \overrightarrow{\mathbf{v_1}}/\overrightarrow{\mathbf{v_3}} & \text{if } c_j^Y = \mathbf{v_2}/\overrightarrow{\mathbf{v_3}} \\ \overrightarrow{\mathbf{v_1}}\backslash\overrightarrow{\mathbf{v_3}} & \text{if } c_j^Y = \mathbf{v_2}\backslash\overrightarrow{\mathbf{v_3}} \\ \overrightarrow{\mathbf{v_1}} & \text{if } c_j^Y = \mathbf{v_2}. \end{cases}$$

and add c_{new} to $actY$'s possible categories in the lexicon.

We will provide an example of the use of these rules after discussing the changes to the probabilities, but some discussion of these rules will help with understanding them. First, note that neither of these rules changes the entry in the lexicon for $actX$. This is necessary to maintain the yield of the grammar for all of the cases in which the action is observed.

Rule one replaces, in every category of the grammar that makes use of it, the causal structure normally provided by the categories of the partially observable action. For example, suppose $actX := \{((A/\{B\})\backslash\{C\}\}$ and we want to make it partially observable. Rule one creates new categories that replace all of the occurrences of A in the lexicon with the structure required for an unobserved instance of $actX$ to achieve A. In this case, the rule would replace instances of A with leftward looking C and rightward looking B arguments.

Note rule one preserves the directionality of the slashes of the original category structure with the definition of c_{new}. This is why there are four cases in its definition. These four cases guarantee that the plan's ordering constraints, encoded in the original grammar, are not violated in the new lexicon.

As a final note about rule one, this rule is formulated as if each atomic category can occur only a single time in any category definition. However nothing guarantees this and we must address this limitation. Rule one must be invoked within a loop to address the addition of multiple possible c_{new} categories. Consider making action $actX$ partially observable and suppose a category A that is a root result of one of its categories, c_i^X, occurs n times as an argument in another category, c_j^Y, in the definition of action $actY$. For completeness the new grammar must consider all of the possible permutations of $actX$ being observed and not in any instance of c_j^Y. This requires the adding to $actY$'s definition of $2^n - 1$ categories to capture all of the possible non-empty instances of A being accomplished by an execution of $actX$ and being observed or not. We will discuss the impact of this on runtime shortly.

Rule number two addresses the possibility that the action to be made partially observable, $actX$, is the anchor for a category, G, that never occurs as an argument in another category. Effectively such an action would be the anchor for what we might think of as a "top level goal" of the agent. If this is the case, to make the action partially observable we have to provide another anchor for the root result in cases where $actX$ is performed but not observed.

Rule two does this by selecting the first argument of the G anchored category, A, and elevating all of the actions that achieve this category to be anchors for G. It does this by constructing new categories that achieve G and adding them to the actions' definitions. Note that it must do this for all of the actions that can achieve A otherwise the grammar will loose elements of the yield where an unobserved execution of $actX$ could have resulted in G.

Modifying the Grammar's Probabilities

As we have already discussed, ELEXIR action definitions specify a probability distribution over the categories that each action can initially take on. However, in both of the yield extending rules above a *source category* is extended to create a new category that is then added to an action's definition. The probability distribution for this action must be changed to accurately account for the new categories. Making an action, $actX$, partially observable will depend on having a false negative rate, r_X for its observation.

For rule number one, the intuition is to allocate some of the probability mass from the source category to the new category on the basis of the false negative rate. If the probability of c_j^Y in the initial grammar is P_j^Y and c_{new} is a category derived from c_j^Y in the process of making $actX$ partially observable using rule one, with a false negative rate of r_X, then in the new lexicon we change the probability of c_j^Y to be $P_j^Y(1 - r_X)$ and define the probability of c_{new} as $P_j^Y r_X$. This is just the original probability of the category times the probability that $actX$ is going to be executed and not observed.

However, as we have pointed out, c_j^Y could have multiple instances of a category A as an argument, and each could be achieved using unobserved executions of $actX$. Therefore we define the probability of a new category more generally as:

$$P_{c_{new}} = \begin{cases} P_j^Y r_X & \text{if } n = 1 \\ P_j^Y (r_X^k - r_X^{k+1})/\binom{n}{k} & \text{if } n > 1 \text{and } k < n \\ P_j^Y r_X^n & \text{if } n > 1 \text{and } n = k. \end{cases}$$

Where n is the number of instances of A in c_j^Y, and k is the number of these instances in c_{new} that will be accounted for by unobserved instances of $actX$.

The second line of this definition requires explanation. The probability mass for all of the categories that result from modifying c_j^Y (including itself) must sum to P_j^Y. We also know that the probability mass associated with all of the new categories that have n or more unobserved actions must sum to $P_j^Y(r_X)^n$. We therefore know the sum of the probability mass of the new categories that have exactly $k < n$ unobserved actions must sum to $P_j^Y((r_X)^k - (r_Y)^{k+1})$ which is the numerator of the second case. We also know there are $\binom{n}{k}$ categories that have exactly k unobserved actions that this probability mass must be divided over. Having no other information about the relative likelihood of the new categories, we distribute the mass over the $\binom{n}{k}$ categories uniformly.

In the case of the categories that are added by the second rule, we are again adding a category that is a modification of an existing category. This means that again it makes sense to divide the probability mass of the source category in order to assign probability mass to the new category.

Remember that, in rule two, the source category's root result is A. In the initial grammar, with some likelihood, $P_{j,Y,X}$, any instance of A, produced by the source category,

could play the role of being the first argument for c_i^X. Let P_j^Y be the probability assigned to the source category in the initial grammar, we know $P_{j,Y,X} < P_i^Y$ because this is a subset of all of the instances of the source category. Thus, in the new grammar we would like to assign the probability $(P_j^Y - P_{j,Y,X}) + P_{j,Y,X}(1 - r_X)$ to the source category and $P_{j,Y,X}(r_X)$ to the new category. This correctly captures the intuition that only the percentage of instances of the source category that would have been used in conjunction with an instance of the unobserved action should be effected by this change.

However, there is no way to determine $P_{j,Y,X}$ in general. It is not given in the grammar and cannot be reconstructed from the probabilities in it. Therefore, again assuming uniformity, we will split the probability mass giving half to the use for the new category and half for the source category. Thus, in the new grammar, we define $P_j^Y = (0.5P_j^Y) + 0.5P_j^Y(1 - r_X)$ and $P_c new = 0.5P_j^Y r_X$.

Note the two uses of a uniform distribution in this discussion are an initial approximation. There are a number of more complex possibilities, including conditioning on state or the parse. We also believe these distributions are learnable from real world data, however these more complex models and questions we leave as an area for future work.

Example

Consider CCG: 2 which captures part of a computer network security domain. This domain has two primary goals: data theft (DT) and denial of service (DOS). The following example, will show the results of rewriting of CCG: 2 making two different actions partially observable. We will show the probability distribution over the categories that is required by the lexicon as a sequence of reals after the category definition of the action.

CCG: 2

$$
\begin{aligned}
portscan &:= \{\ S\ \}[1.0] \\
remote2loc &:= \{\ (((DT/\{DX\})/\{C\})/\{U2R\})\backslash\{S\}\ \}[1.0] \\
usr2root &:= \{\ U2R\ \}[1.0] \\
consolidate &:= \{\ C\ \}[1.0] \\
dataex &:= \{\ DX\ \}[1.0] \\
synflood &:= \{\ (DOS)\backslash\{S\}\ \}[1.0]
\end{aligned}
$$

Making the action $usr2root$ partially observable with a false negative rate of 0.25 results in the grammar shown in CCG: 3. Only rule number one is applied in this case because the action in question is not the anchor of a top level category. Note however, this does result in the addition of a category to the definition of action remote2loc and the changing of the probability distribution over its categories.

Next, we again apply the algorithms making the action $synflood$ partially observable with a false negative rate of 0.25. This results in CCG: 4. Rule two is applied here because $synflood$ is the anchor for a top level category DOS. Therefore, we add a new category to action $portscan$, making it a new anchor for DOS when $synflood$ is not observed, and modifying the distribution over its categories.

CCG: 3

$$
\begin{aligned}
portscan &:= \{\ S\ \}[1.0] \\
remote2loc &:= \{\ (((DT/\{DX\})/\{C\})/\{U2R\})\backslash\{S\}, \\
&\qquad \{\ ((DT/\{DX\})/\{C\})\backslash\{S\}\ \}[0.75, 0.25] \\
usr2root &:= \{\ U2R\ \}[1.0] \\
consolidate &:= \{\ C\ \}[1.0] \\
dataex &:= \{\ DX\ \}[1.0] \\
synflood &:= \{\ (DOS)\backslash\{S\}\ \}[1.0]
\end{aligned}
$$

Since the rewriting of grammars will be done offline, before plan recognition, it is not necessary that it be an efficient process, and therefore we will not discuss its complexity.

CCG: 4

$$
\begin{aligned}
portscan &:= \{\ S, DOS\ \}[0.875, 0.125] \\
remote2loc &:= \{\ (((DT/\{DX\})/\{C\})/\{U2R\})\backslash\{S\}, \\
&\qquad \{\ ((DT/\{DX\})/\{C\})\backslash\{S\}\ \}[0.75, 0.25] \\
usr2root &:= \{\ U2R\ \}[1.0] \\
consolidate &:= \{\ C\ \}[1.0] \\
dataex &:= \{\ DX\ \}[1.0] \\
synflood &:= \{\ (DOS)\backslash\{S\}\ \}[1.0]
\end{aligned}
$$

Impact and Evaluation

To understand the impact grammar rewriting has on runtime, it will be helpful to consider the impact it has on the yield of the grammar in general. consider a sequence of actions of length m that is in the yield of the grammar and has n instances of an action, $actX$, that is to be made partially observable. As we talked about, for yield extension rule one, any new grammar that allows $actX$ to be partially observable, must accept at least $2^n - 1$ new action sequences, one for each possible non-empty subset of the n actions being unobserved. Thus, addressing partially observable actions requires the yield of the plan grammar to increase.

To achieve this, rewriting of the grammar can add, in the worst case, $2^n - 1$ categories to an individual action's definition, where n is the number of instances of a selected argument in the source category. This can increase the branching factor of the search for explanations and therefore the system's runtime, since the number of possible explanations is the largest determiner of ELEXIR's runtime (Geib 2009). However three factors tend to minimize this.

First, in practice, n is usually small. Remember, n is the maximum number of times the same category occurs as an argument to a category. This is comparable to the number of times a subgoal recurs in a single decompositional method in a hierarchical plan(Erol, Hendler, and Nau 1994). As with such hierarchical plans, where the number of subgoals in a method is usually in the single digits, in our experience it is very unusual to have a category with more than ten arguments, and n must be less than this maximum. Thus, again, in our experience values of n larger than three are very unusual making the impact on the branching factor small.

	No PO	PO present	PO missing
init	1.18 (0.3) [19]	3.24 (0.5) [53]	1.90 (0.1) [24]
mod	0.89 (0.1) [19]	2.53 (0.5) [79]	1.61 (0.5) [37]

Figure 2: Runtimes in milliseconds, standard deviation (in parens), and number of explanations (in square brackets)

Second, the branching factor of the entire grammar is not increased by $2^n - 1$. This increase in the branching factor is localized to those explanations that actually make use of actions that have had categories added to them and only to those parts of the plan where the action is used. Thus the $2^n - 1$ increase in the branching factor is a localized effect.

Third, the runtime of the ELEXIR algorithm is most closely tied to the number of possible explanations for a given set of observations (rather than the size of the grammar) (Geib 2009). Thus, an increase in the grammar size and yield may not translate into an increase in the number of explanations for a particular sequence of observations. While the size and yield of the grammar may go up, the number of possible explanations for a given set of observations may go up or stay the same. As a result, establishing the efficacy of this method may require empirical evaluation.

For example, Figure 2 shows average runtimes in milliseconds for ten executions in each of the six test conditions using both the initial grammar and a grammar modified to deal with a single partially observable action. The domain has twenty three possible goals (high level plans), nineteen observable actions, and is based on a computer security domain taken from the DARPA MRC project. Each input was the first five steps of a larger plan (this was done in order to remove any effects from varying lengths of observation sequences). The runtimes, standard deviations (in parentheses), and the number of explanations produced (in square brackets) are shown for three recognition test cases: column 1, the observed plan that has no partially observable actions in it (though such actions are present in the domain), column 2, a plan that has partially observable actions and their observations are present in the input observations, and column 3, a plan that has partially observable actions with missing observations. The test cases for the third case were generated from the test cases for the second case by removing the partially observable action and adding the next action in the plan to the observation stream.

We include the first test case (column one), to show that adding partially observable actions need not have a significant impact on recognition of plans that don't involve the partially observable action. This is attested by the negligible differences in runtime between the two cases, and the fact that the same explanations are generated. This is a particularly nice result as adding partially observable actions to other methods like (Geib and Goldman 2005) will increase the runtime of the recognition algorithm across the board.

The most interesting finding in the other cases is that the runtimes with the partial observed actions missing are faster than those where in the actions are observed. This is a result of the smaller number of possible explanations. In the case where the actions are observed, the system still has to consider the possibility that the observed action is part of some other plan resulting in a larger number of explanations.

Note the runtimes in the upper third column for the initial grammar are not strictly comparable to the others since ELEXIR fails to find the true plan as a result of the missing observations in the initial grammar. While ELEXIR doesn't find the actual plan, it does produce twenty four reasonable explanations for the observed actions accounting for the runtime being larger than in the modified case.

Conclusions

The contribution of this work is in demonstrating the viability of automatically rewriting plan grammars to address partial observability in plan recognition. We have done this and shown that a particular system, the ELEXIR system, can effectively use such modified plan grammars with limited impact. However, the ideas discussed in this work could be applied to a number of other plan recognition systems, namely any of those that use an explicit plan library. While a longitudinal study of the efficacy of this method across all other systems, and the features of those systems that could make its use problematic, would be interesting, we leave this for future work.

Handling partially observable actions in this way, required no algorithmic changes for the ELEXIR system, only a recoding of the domain. That said, handling such partially observable actions requires an increase in the yield of the grammar, and as a consequence, in some cases we can see an increase in the runtimes for plan recognition. However, because ELEXIR uses a bottom up, incremental parsing algorithm, the largest increases in runtime for it occurs when partially observable actions are actually observed.

Note that in developing our algorithm, no effort was made to produce the optimal grammar. It is well noted that for almost all grammar formalisms there are multiple grammars with the same yield. Further, Geib (Geib 2009) has noted that some CCG plan lexicons result in much faster parsing. We have left optimizing the rewritten grammar as an area for future work, and there is the potential that significant reductions in runtime may be possible in such grammars.

Acknowledgements

The author would like to thank Robert Goldman and Mark Burstine for helpful discussions on the ideas presented in this paper. This work was supported by Contract FA8650-11-C-7191 with the US Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory. Approved for Public Release, Distribution Unlimited. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2005. Fast and complete symbolic plan recognition. In *Proceedings of IJCAI*, 653–658.

Bui, H. H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the Abstract Hidden Markov Model. *Journal of Artificial Intelligence Research* 17:451–499.

Carberry, S. 1990. *Plan Recognition in Natural Language Dialogue*. ACL-MIT Press Series in Natural Language Processing. Cambridge, MA: MIT Press.

Curry, H. 1977. *Foundations of Mathematical Logic*. Dover Publications Inc.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task network planning. In *Proceedings of AIPS*, 249–254.

Geib, C. W., and Goldman, R. P. 2005. Partial observability and probabilistic plan/goal recognition. In *In Proceedings of the 2005 International Workshop on Modeling Others from Observations (MOO 2005)*.

Geib, C., and Goldman, R. 2011. Recognizing plans with loops represented in a lexicalized grammar. In *Proceedings of AAAI*, 958–963.

Geib, C., and Steedman, M. 2007. On natural language processing and plan recognition. In *Proceedings of IJCAI 2007*, 1612–1617.

Geib, C. 2009. Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. In *Proceedings of IJCAI*, 1702–1707.

Hoogs, A., and Perera, A. A. 2008. Video activity recognition in the real world. In *Proceedings of AAAI*, 1551–1554.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley.

Kabanza, F.; Filion, J.; Benaskeur, A. R.; and Irandoust, H. 2013. Controlling the hypothesis space in probabilistic plan recognition. In *Proceedings of IJCAI*, 2306–2312.

Kautz, H. A. 1991. A formal theory of plan recognition and its implementation. In Allen, J. F.; Kautz, H. A.; Pelavin, R. N.; and Tenenberg, J. D., eds., *Reasoning About Plans*. Morgan Kaufmann. chapter 2.

Liao, L.; Fox, D.; and Kautz, H. A. 2005. Location-based activity recognition using relational Markov networks. In *Proceedings of IJCAI*, 773–778.

Litman, D. 1986. Understanding plan ellipsis. In *Proceedings of AAAI*, 619–626.

Pynadath, D., and Wellman, M. 2000. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of UAI*, 507–514.

Ramirez, M., and Geffner, H. 2011. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *Proceedings of IJCAI*, 2009–2014.

Steedman, M. 2000. *The Syntactic Process*. MIT Press.

Vail, D. L., and Veloso, M. M. 2008. Feature selection for activity recognition in multi-robot domains. In *Proceedings of AAAI*, 1415–1420.

Vilain, M. B. 1990. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proceedings of AAAI*, 190–197.

Vilain, M. 1991. Deduction as parsing. In *Proceedings of AAAI*, 464–470.