Recognizing Plans with Loops Represented in a Lexicalized Grammar.

Christopher W. Geib
School of Informatics
University of Edinburgh
10 Crichton Street,
Edinburgh, EH8 9AB, Scotland
cgeib@inf.ed.ac.uk

Robert P. Goldman,
SIFT, LLC
211 North First Street, Suite 300
Minneapolis, MN 55401-1480
rpgoldman@sift.info

Abstract

This paper extends existing plan recognition research to handle plans containing loops. We supply an encoding of plans with loops for recognition, based on techniques used to parse *lexicalized* grammars, and demonstrate its effectiveness empirically. To do this, the paper first shows how encoding plan libraries as context free grammars permits the application of standard rewriting techniques to remove left recursion and ϵ-productions, thereby enabling polynomial time parsing. However, these techniques alone fail to provide efficient algorithms for plan recognition. We show how the loop-handling methods from formal grammars can be extended to the more general plan recognition problem and provide a method for encoding loops in an existing plan recognition system that scales linearly in the number of loop iterations.

Introduction

Vilain (1990) showed that plan recognition (the problem of inferring which plan, from a known set of possible plans, an agent is executing based on observations of their actions) is NP-hard for most interesting classes of plan libraries. Vilain did this by reducing the problem of plan recognition to parsing. He also proved the existence of a polynomial time algorithm for the restricted class of hierarchical plans (with loops) that are ordered and do not share plan steps.

Subsequent work has made efforts to provide empirically effective algorithms that remove some or all of the limitations Vilain required to get the polynomial time result. In so doing, they have often moved away from representing their plans as formal grammars. Unfortunately, many (Kautz 1991; Avrahami-Zilberbrand and Kaminka 2005; Geib, Goldman, and Maraist 2008) of these alternative representations and systems are relatively impoverished and do not fully support looping plans.

Even work that has suggested using plan grammars (Geib, Goldman, and Maraist 2008; Pynadath and Wellman 2000) has failed to make use of the full expressivity of the grammatical models. Formal language theory provides a method for rewriting plan grammars to enable recognition of loops with no additional machinery beyond existing parsing algorithms. In much of the prior work, this result has been lost. This paper reintroduces these methods to plan recognition.

However, given Vilain's results, we must expect that any system that does allow for recognition of multiple goals and partially ordered plans must be at least NP-hard. Therefore, it will not be enough to simply apply past parsing results. Instead, we modify these techniques for efficient use in an existing plan recognition system that is capable of handling partial orders and multiple, interleaved goals. The result is an efficient plan recognition system incorporating a novel loop encoding.

The rest of this paper has the following structure: First, we discuss previous work in plan recognition and the problem of recognizing plans with loops. Second, we discuss loop rewriting techniques for formal plan grammars. Third, we briefly review ELEXIR, a probabilistic plan recognition system based on parsing *lexicalized grammars*. Finally we provide an encoding for plans with loops in ELEXIR that scales linearly with the number of loop iterations, and provide empirical evidence for its efficiency.

Prior Work

Graph-based algorithms have been at the core of a number of pieces of previous research. Some of these, (Kautz 1991; Avrahami-Zilberbrand and Kaminka 2005), did not explicitly discuss the problem of recognizing plans with loops. However systems that view plan recognition as graph covering or that involve marking elements of a graph require modifications to handle looping plans. The graph models for such algorithms must be extended to model the loops. A naive implementation requires a bound on the number of iterations of each loop and a separate graph for each possible number of iterations. More sophisticated implementations are possible, but we know of no such reported work.

CRFs and HMMs have been the foundation of much of the work in activity recognition. This approach is based on discretizing the state space into regions, modeling the transitions between the regions with Hidden Markov Models(HMM) (Bui, Venkatesh, and West 2002) or Conditional Random Fields (CRF) (Hoogs and Perera 2008; Liao, Fox, and Kautz 2005; Vail and Veloso 2008) and then using the HMM or CRF to determine the most likely plan being followed. This approach has subtle problems for trajectories that enter the same region more than once. In such a situation, the Markov assumption can mean that the system is effectively unaware that it has been in the region. Thus,

without additional processing, such a plan recognizer will be unable to tell how many times it had been around a loop. Losing this information is what restricts the grammars supported by HMMs to be strictly regular. This can have both an impact on the probability model (e.g. the probability of continuing a loop may depend on how many times the loop has already been executed) and even on the ability to determine if a plan is well formed (e.g. imagine that the number of times a loop is executed distinguishes two plans).

Representing Plans as Grammars and treating plan recognition as parsing was critical for Vilain's (1990) proofs of the complexity of the plan recognition problem. However, his work does not present an algorithm or implemented system for plan recognition.

Pynadath and Wellman (2000) provide a method for plan recognition based on probabilistic context free grammars (PCFGs). They do not directly parse the observations to produce derivations. Instead, they use the structure of plans captured in a PCFG to build a restricted Dynamic Bayes Net (DBN) to model the underlying generative process of plan construction and execution. They use the resulting DBN to compute a distribution over the space of possible plan expansions that could be currently under execution. In order to build their DBN, they are only able to handle a limited class of loops. Further their treatment of loops ignores the number of iterations of the loop. So like the work on HMMs and CRFs, they can recognize loops but not capture how many times the loop has been executed.

Some more recent work (Geib, Goldman, and Maraist 2008; Geib and Goldman 2009) has directly used formal grammars to represent plans and modified parsing algorithms to perform plan recognition. Unfortunately these algorithms have not made full use of previous results from formal grammar theory. These systems perform bottom-up parsing of grammars capturing the leftmost derivation trees of an underlying context free plan grammar representing the Hierarchical Task Network (HTN) (Ghallab, Nau, and Traverso 2004) plans to be recognized. In order to construct a finite set of such trees, this work bounded the maximum number of times the agent could execute a loop. However as we will see next, this limitation is unnecessary.

Plans as Grammars

Limited forms of HTN plans are often used to define the plan library for recognition. Such plans can be straightforwardly represented as CFGs (Erol, Hendler, and Nau 1994). Consider the simple plans for going to a conference (GO2CON), including traveling to a location (T2L), shown in Figure 1 and encoded in CFG:1

CFG: 1

$$
\begin{aligned}
GO2CON &\rightarrow PACK, T2L, CHECKIN \\
PACK &\rightarrow pack \\
T2L &\rightarrow W \mid T2L, TT, W \\
TT &\rightarrow ride \\
CHECKIN &\rightarrow talk2C \\
W &\rightarrow walk
\end{aligned}
$$

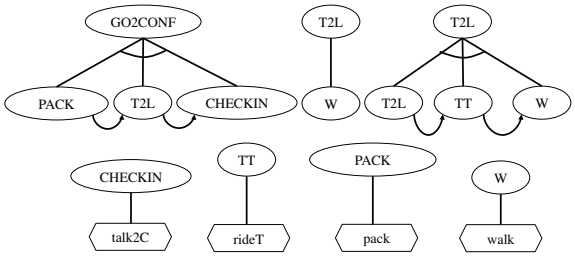

Figure 1: A simple set of hierarchical plans.

Generally, parsing systems accept as input a *complete* syntactic unit (a sentence, a program, procedure definition, etc.), verify that it is well-formed, and yield a data structure (such as a parse tree) representing the syntactic analysis. However plan recognition presents a subtly different task. In many, if not most, application domains for plan recognition, we must be able to recognize the plan being performed *before* it is completed. For example, an assistive system might wish to recognize the plan a user has in progress so that it can help the user complete that plan. The equivalent in parsing is incremental parsing which might be done in an IDE[1] to help a programmer ensure a well-formed program, or in natural language processing to permit interleaving syntactic and semantic inferences. Unfortunately, grammars like CFG:1, above, are problematic for simple incremental parsing algorithms.

The second production for T2L (this is the production that captures the loop for taking multiple trains) is *left recursive*. The first element of the right hand side of the production is the same as the non-terminal on the left hand side. Whenever the parser reaches such a left recursive production it can no longer be driven only by explaining the observation. It must search the space of possible parses to determine how many times this rule will be applied. This amounts to searching to determine how many times the loop is executed.

Many existing parsing algorithms implicitly use the length of the input sequence and the assumption of a complete sentence to bound this search. However, for incremental algorithms this is not possible. It was to limit this search that (Geib, Goldman, and Maraist 2008; Geib and Goldman 2009) bounded the number of times a loop could be executed. However, formal language theory has shown that left recursion can be removed from a CFG (Hopcroft and Ullman 1979) without bounding looping constructs.

Removing Left Recursion

Removing left recursion from a grammar is based on the realization that any left recursive production can be changed into a rightward recursive rule and an ϵ-production (a production that has an empty right hand side). Thus, a production of the form: $A \rightarrow A\gamma \mid B$ can be rewritten as the two

[1] Integrated Development Environment

productions: $A \rightarrow B \mid BX; X \rightarrow \epsilon \mid \gamma X$[2]. Following the algorithm for this rewriting process provided by Hopcroft and Ullman (1979), CFG:1 can be converted to the following equivalent grammar:

CFG: 2

$$\begin{aligned} GO2CON &\rightarrow PACK, T2L, CHECKIN \\ PACK &\rightarrow pack \\ T2L &\rightarrow W \mid WX \\ X &\rightarrow \epsilon \mid TT, W, X \\ TT &\rightarrow ride \\ CHECKIN &\rightarrow talk2C \\ W &\rightarrow walk \end{aligned}$$

that does not have left recursion. Note the introduction of the new non-terminal X encoding the loop. This allows the grammar to be written as right, rather than left, recursive.

As we see in our example, the removal of left recursion can introduce ϵ-productions into the CFG and these can complicate parsing as well. However, there is also an algorithm for removing ϵ-productions from a grammar. The intuition behind ϵ-production removal is "multiplying out" all the possible alternatives. The end result is that we add a production for each possible case. We again refer the interested reader to (Hopcroft and Ullman 1979) for more detail. ϵ-production removal from our grammar yields:

CFG: 3

$$\begin{aligned} GO2CON &\rightarrow PACK, T2L, CHECKIN \\ PACK &\rightarrow pack \\ T2L &\rightarrow W \mid WX \\ X &\rightarrow TT, W \mid TT, W, X \\ TT &\rightarrow ride \\ CHECKIN &\rightarrow talk2C \\ W &\rightarrow walk \end{aligned}$$

The critical change here is in the production for X. [3]

The removal of left recursion and epsilon productions results in a CFG with a finite number of productions suitable for polynomial time parsing. Thus, from a strictly theoretical view point, for plan recognition systems where the plans can be expressed as a CFGs, there is no reason to limit or bound plans with loops.

However, this theoretical result does not end the discussion. We must ask how effective using grammatical parsing for recognition of plans with loops is in practice. This immediately raises two significant problems. First, as we have noted, the guarantee of polynomial runtime for CFG parsing is based on assuming that we have a complete trace of a single plan. We cannot make these assumptions in most plan recognition applications. Second, even if we could show that incremental parsing for partial traces with multiple interleaved goals is as efficient,[4] plans with loops are likely to have much longer traces and polynomial time algorithms may be insufficient.

Thus, we need to find efficient encodings of loops in incremental plan recognition systems that support multiple interleaved plans. The next section will briefly review such an existing, grammar based recognition system. We will then discuss a specific method of encoding plans with loops in the system and verify its effectiveness with empirical results.

System Background on ELEXIR

ELEXIR(Geib 2009), is a probabilistic plan recognition system based on Combinatory Categorial Grammars (CCG) (Steedman 2000). As we have required above, ELEXIR goes beyond conventional parsing to incrementally recognize multiple, interleaved plans. Due to space constraints we can only provide an overview of ELEXIR here, and refer the reader to (Geib 2009) for more details.

Following current research trends in natural language processing, the CCGs ELEXIR uses to represent plans are a form of *lexicalized grammar*. In traditional CFGs, constraints specific to a language are spread between the rules of the grammar and the lexicon. Lexicalized grammars move all language-specific information into the lexicon in the form of "lexical categories" and use a small set of language independent rules to combine the lexical entries. Parsing lexicalized grammars abandons the application of multiple grammar rules in favor of assigning a lexical category to each observation and combining the categories to build a parse.

Thus, to represent a plan library in a CCG, each observable action is associated with a set of syntactic *categories*. The set of possible categories, C, is defined recursively as:

Atomic categories : A finite set of basic action categories. $C = \{A, B, ...\}$.

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* ($\{W, X, ...\}$) and produce a *result* (Z). The direction of the slash indicates where the function looks for its arguments. We require the argument(s) to a complex category be observed after the category for forward slash, or before it for backslash in order to produce the result. Continuing our conference traveling example, one possible encoding of CFG:3 in a CCG lexicon is:

CCG: 1

$$\begin{aligned} pack &:= (GO2CON/\{CHECKIN\})/\{T2L\}. \\ walk &:= W \mid T2L \mid T2L/\{X\}. \\ ride &:= X/\{W\} \mid (X/\{X\})/\{W\}. \\ talk2C &:= CHECKIN. \end{aligned}$$

[2]Where ϵ is the empty string.

[3]Note that in general removing ϵ-productions requires a special case to handle grammars that can yield the empty string, but such grammars are typically not interesting for plan recognition.

[4]This seems unlikely given that recognizing multiple interleaved goals requires recognizing which plan an observed action participates in even before plan recognition can be done.

Where the loop body is encoded in the categories for *ride*.

CCG categories are combined into higher level plan structures using *combinators* (Curry 1977). ELEXIR only uses three combinators defined on pairs of categories:

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

where X and Y are categories, and α and β are possibly empty sets of categories. To see how a lexicon and combinators parse observations into high level plans, consider the derivation in Figure 2 that parses the observation sequence: *pack*, *walk*, *talk*2*C* (the very simple plan for walking to the conference without taking a train) using CCG:1. As each

$$\dfrac{\dfrac{\overline{(GO2CON/\{CHECKIN\})/\{T2L\}}\ \overline{T2L}}{GO2CON/\{CHECKIN\}}>\quad \overline{CHECKIN}}{GO2CON}>$$

pack walk talk2C

Figure 2: Parsing Observations with CCGs

observation is encountered, it is assigned a category as defined in the lexicon. Combinators (rightward application in this case) then combine the categories.

To enable incremental parsing of multiple interleaved plans, ELEXIR does not use an existing parsing algorithm. Instead it uses a very simple two step algorithm based on combinator application linked to the in-order processing of each observation.

- Step 1: Before an observation can be added to the explanation with a given category, it must be possible to use leftward application to remove all of the category's leftward looking arguments. We call such categories *applicable* given the observed action and current explanation.
- Step 2: After each of the applicable categories for the observation has been added to a copy of the explanation, ELEXIR applies all possible single combinators to each pairing of the new category with an existing category.

This two step algorithm both restricts observations to take on only applicable categories, and guarantees that all possible combinators are applied. At the same time, it does not force unnecessarily eager composition of observations. For example, given the observations *pack*, *walk*, *talk*2*C*, ELEXIR produces three explanations:

$[GO2CON]$,

$[GO2CON/\{CHECKIN\}, CHECKIN]$, *and*

$[(GO2CON/\{CHECKIN\})/\{T2L\}, T2L, CHECKIN]$.

Each of these parses results from *not* applying one of the possible combinators (in this case rightward application) to an existing explanation when an observation was initially added. The second two parses are included precisely in the case that later observations require these categories to discharge their leftward arguments. More details on why this is necessary can be found in (Geib 2009).

Representing Loops in ELEXIR

Representing loops in a CCG is relatively straightforward. All that is required is for our lexicon to include at least one category that has an argument that has the same category as its leftmost result. For example, in CCG:1 the lexical entry:

$$ride := (X/\{X\})/\{W\}$$

says that observing a *ride* is a function that produces an X when its executed in a context where W is observed next, but only if it is followed by another instance of X, hence a loop. For this to be useful, there must be another lexical entry that produces an X to terminate the loop. In CCG:1, the lexical entry for *ride* fulfills both of these functions. It can take on the category that encodes a loop ($(X/\{X\})/\{W\}$) or a simpler category that results in an X without a further loop ($X/\{W\}$).

Unfortunately, this simple encoding of loops comes into conflict with ELEXIR's algorithm. In an effort to leave categories available for use as leftward arguments of future observations, ELEXIR can create explanations that leave iterations of the loop body as separate goals. For example, given observations: *pack*, *walk*, *ride*, *walk*, *talk*2*C* in addition to producing an explanation of these observations as a single instance of *GO2CON*, ELEXIR will produce a number of other explanations including:

$$[GO2CON,\ X/\{X\}]$$

where the instance of *GO2CON* is explained by the *pack*, the first *walk* and the *talk*2*C* observations, and $X/\{X\}$ is explained by the *ride* and the second *walk* observations. However, given our knowledge of the loop and the grammar, this explanation makes no sense. The non-terminal X was added to the grammar to function as part of a loop for other plans not as a standalone goal. Worse yet, for each iteration of the loop the system is able to generate another instance of $X/\{X\}$, along with other longer, similarly nonsensical explanations for the observations. Note that this problem is not unique to ELEXIR. Any incremental parsing algorithm will face this problem as well.

Instead of this, we would like the system to leverage what we know about the loop, namely that this loop works in service of a higher goal and is not a viable plan the agent can be performing for its own sake. We would like the system to only maintain two explanations: one that assumes the loop terminates on the next observation, and one that assumes that the loop continues. We have identified an encoding of loops in CCG that forces the system to do this based on the use of complex categories as arguments.

Encoding Loops for Efficiency

Forcing ELEXIR to consider the loop as a single unit requires a subtle representational trick. We need to explicitly represent the structure of the plan as having actions that come before the loop, actions that are within the loop, and actions that come after the loop. Suppose we want to recognize sequences where we have an instance of *act*1 (the *loop prefix*) followed by some number of instances of *act*2 (the *loop body*) followed by a single *act*3 (the *loop suffix*). Imagine encoding this in the following CCG.

CCG: 2

$$act1 := PRE.$$
$$act2 := (G/\{POST\})\backslash\{PRE\} \mid (G/\{G\}.$$
$$act3 := POST.$$

However, as we have seen, because of the inclusion of the $G/\{G\}$ category, this lexicon will result in an ever increasing number of explanations as the loop is executed.

To prevent this, we can define the category for looping with a complex category as an argument and force the system to treat the loop prefix, body, and suffix as a single unit. For example, in the following lexicon:

CCG: 3

$$act1 := PRE.$$
$$act2 := (G/\{POS\})\backslash\{PRE\} \mid (G/\{POS\})\backslash\{G/\{POS\}\}.$$
$$act3 := POS.$$

the complex category argument for action *act*2's second category requires that a previous instance of the loop body (that itself was preceded by the loop prefix) has already been observed before another instance of *act*2 can be recognized as part of the loop. Discharging the complex argument results in a $G/\{POS\}$ which can be composed with another *act*2, or can have an instance of *act*3 applied to it to complete an instance of the plan. Thus $G/\{POS\}$ acts as an efficient representation of both the possibility that the loop continues and that it terminates on the next action. An example of this parsing process in shown in Figure 3.

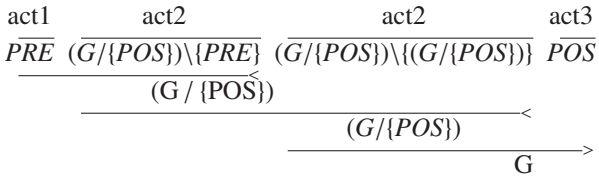

Figure 3: Efficient Loop Parsing with CCGs

Using this encoding, our example domain is captured in the following lexicon:

CCG: 4

$$pack := PACK.$$
$$walk := W \mid PRE\backslash\{PACK\}.$$
$$ride := ((GO2CON/\{POS\})/\{W\})\backslash\{PRE\} \mid ((GO2CON/\{POS\})/\{W\})\backslash\{(GO2CON/\{POS\})\}.$$
$$talk2C := POS.$$

Keep in mind that in the case of our example plan grammar, the loop has a body that contains two actions, namely: *walk* and *ride*. To account for this we have explicitly included an argument in the looping categories for the walking action.

This encoding forces the algorithm to consider the loop and its prefix and suffix as one entity. This means that it only maintains a very small number of open hypotheses and can represent and recognize long plans very efficiently. Further, it does this without making any changes to the ELEXIR algorithm itself.

	3	5	7	9	12
CCG:1	2640	-	-	-	-
CCG:5	2.0	68	51351	-	-
CCG:4	0.4	0.6	0.8	0.9	1.3

Figure 4: Runtime in msec. for three CCG encodings of loops versus number of loop iterations.

Empirical Results

To test the efficiency of this encoding of loops, we have run a number of tests for our example domain using both CCG:4 and CCG:1 varying the number of loop iterations with a one minute bound on the runtime. In the interest of completeness we have also tested a third grammar encoding the domain:

CCG: 5

$$pack := PACK.$$
$$walk := W \mid T2L\backslash\{X\} \mid T2L\backslash T2L.$$
$$ride := X.$$
$$talk2C := ((GO2CON\backslash PACK)\backslash W)\backslash T2L.$$

Unlike CCG:1 which uses all rightward slashes, CCG:5 uses leftward slashes in all of its complex categories. (Geib 2009) points out that such leftward looking grammars effectively delay commitment to high level goals and therefore have much lower runtimes.

The results for all three grammars are reported in milliseconds in Figure 4. Dashes mark tests that did not return within the one minute bound. The results clearly show that encoding loops using a complex category as an argument results in exceptional runtime savings. It beats both of the alternative simpler encodings by orders of magnitude. It shows the new encoding allows the recognition of plans that are not within the runtime bound of the other encodings. It also verifies that this encoding makes the runtime for recognizing loops linear in the number of loop iterations.

This approach does require a prefix, body, and suffix for the loop. Without the prefixes and suffixes to bound the loop this approach is not viable. Thus, efficiently recognizing loops in this way may not always be possible and can require knowledge engineering during lexicon construction.

The Probability Model

ELEXIR is a probabilistic recognizer, and we do need to discuss its probability model. ELEXIR views probabilistic plan recognition as weighted model counting using parsing for model construction. Since each parse provides a model that "explains" the observed actions, in the following, we will use the term *explanation* in place of "parse" or "model."

To compute the conditional probability for each of the possible root goals, ELEXIR needs to compute the probability of each explanation for the observations. For an explanation, *exp*, of a sequence of observations, $\sigma_1 ... \sigma_n$, that results

in m categories in the explanation, ELEXIR computes the probability of the explanation as:

Definition 1.1

$$P(exp \wedge \{\sigma_1 ... \sigma_n\}) = \prod_{i=1}^{n} P(cinit_i|\sigma_i) \prod_{j=1}^{m} P(root(c_j))K$$

Where $cinit_i$ represents the initial category assigned in this explanation to observation σ_i and $root(c_j)$ represents the root result category of the jth category in the explanation(exp). See (Geib 2009) for more details.

The first term captures the likelihood of each observation being given the lexical category it has been assigned in the parse given the set of possible categories the lexicon allows it to have. This is standard in NLP parsing and assumes the presence of a probability distribution over the possible categories each observation can have in the lexicon. This term has previously been modeled by a uniform distribution (all the alternatives are considered equally likely), but this assumption is *not* enforced by the model.

The second term is the prior probability of the agent having the root goals present in the explanation. This is the probability that the current categories in the explanation are the agent's root goals and will not be combined into a larger goal. This is the term affected by loops. In fact, this term is incorrect without using the new encoding we have provided.

Consider again our example using CCG:1. We have already stated that the loop body is not a possible goal of the agent in its own right. However, we know that CCG:1 can produce a possibly large number of explanations that have the loop body as a root goal. These explanations should have zero probability mass associated with them. However, as it stands ELEXIR does not remove these explanations. Instead it computes these probabilities based on default information, thus throwing off the probability computations for all of the root goals. Using the new encoding of loops, these extraneous explanations are not generated and therefore the computed probabilities for all root goals are correct. Thus, the new encoding for loops actually improves the relationship between explanation generation and the probability model, and results in more accurate probability calculations.

Conclusions

This paper makes three central points. First, viewing plan recognition as parsing of a formal grammar can immediately enable recognition of loops by employing existing techniques for removing left recursion from grammars. Second, an existing plan recognition system, (ELEXIR), based on parsing combinatory categorial grammars (CCGs) avoids some of the limitations of previous research on parsing (e.g. assuming a whole plan is observed, that only one plan is observed, and others), while supporting looping. However representational choices in the grammar can make this inefficient. Third, use of complex arguments to encode loops in CCGs overcomes such inefficiencies by eliminating the possibility of considering the loop body as a goal on its own. Thus, this paper provides a method to efficiently address a significant limitation of previous work, and strengthens the view of plan recognition as parsing lexicalized grammars.

Acknowledgements

Dr. Geib's work in this paper was supported by the EU Cognitive Systems project Xperience (EC-FP7- 270273) funded by the European Commission.

References

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

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.

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 AIPS*, 249–254.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101–1132.

Geib, C. W.; Goldman, R. P.; and Maraist, J. 2008. A new probabilistic plan recognition algorithm based on string rewriting. In *Proceedings ICAPS*, 81–89.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

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

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

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.

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

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 AAAI*, 1415–1420.

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