Handling Looping and Optional Actions in YAPPR.

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

Previous work on the YAPPR plan recognition system provided algorithms for translating conventional HTN plan libraries into lexicalized grammars and treated the problem of plan recognition as one of parsing. To produce these grammars required a fixed bound for any loops within the grammar and a presented a problem for optional actions within HTN plans. In this work we show that well known transformations from formal language theory can be used to rewrite the plan grammars to remove these limitations on the plan libraries.

Introduction

Plan Recognition is generally defined as the problem of inferring which plans, from a given plan library, an agent is executing based on observations of their actions. Many of the representations used for plan recognition are relatively impoverished and do not support complex control structures like looping and optional actions. However, plans that use these structures are common in the real world and therefore we must provide algorithms capable of recognizing them. Some prior work in this area has been based on representing the plans to be recognized with a formal grammar (we will call these *plan grammars*) and using parsing to construct explanations for the observed actions in much the same way that natural languages are parsed for understanding(Carberry 1990; Geib, Goldman, and Maraist 2008; Pynadath and Wellman 2000). However, this work failed to highlight some of the results from previous work in formal grammar rewriting and compiler theory. These results allow a straightforward rewriting of the plan grammars to enable the recognition of loops and optional actions with no additional machinery for the plan recognition task. Therefore, in this paper we will discuss the use of these methods to enable a specific existing plan recognition system to recognize plans with loops and optional actions.

Related Work

Graph Based Methods: Many pieces of previous work (Kautz 1991; Avrahami-Zilberbrand and Kaminka 2005), did not explicitly discuss the problems of recognizing plans with optional actions or loops. However systems that viewed the problem of plan recognition as graph covering or involved marking elements of a graph require modifications to the plan graph to handle optional actions or looping. Such systems will have to include separate elements in the plan graph for loops of arbitrary length. A naive implementation of this would require multiple instances of the same plan with loops of increasing length. Such an approach will have to bound the depth of the loop in order to stop building the plan graph. Such systems will also have to include elements in the plan graph for two different cases for optional actions: where the action is and where it is not. This is particularly problematic if the optional action is reasonably high level and might require duplication of significant portions of the plan graph.

In contrast, the work of (Huber and Simpson 2004) does explicitly discuss loops and recognizing loops. However this work is actually addressing a different problem. It is not actually recognizing human behavior, but rather trying to recognize if looping is happening for script construction.

Grammar Based Methods: Some previous work on grammar based methods (Geib 2006; Geib, Goldman, and Maraist 2008) do explicitly mention recognizing loops and optional actions. However, their solution is to formalize the "unfolding" of the loops we have discussed. While the use of a grammatical formalism means that they are not required to build the complete plan graph before beginning to perform plan recognition, for some cases these systems will be required to unfold the underlying plan. They build grammatical productions for loops of specified lengths. The number of iterations of the loop is bounded and the grammar is unfolded to provide an production for each possible loop length. This is effectively the same as the naive implementation we discussed for graph based methods

Activity Recognition and HMMs: Much of the work in activity recognition is based on discretizing the space into regions, modeling the transitions between the regions with an 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. However this approach has subtle issues with trajectories that crossed over the same part of the state-space multiple times. In such a situation, the Markov assumption can mean that the system is effec-

tively "unaware" that it has even been in this state before. Thus while the steps of the looping plan can be recognized, without additional processing, such a plan recognizer will be unable to tell how many times it had been around a loop. Loosing this information also means that the kind of nested loops that are the hallmark of context free grammars are not supported by HMMs. Distinguishing when a state has been visited would require significant additions to this approach.

Thus, both looping and optional actions have presented problems for previous work in plan recognition. As we will see, basing our plan recognition on parsing will allow us to use results and algorithms from formal grammar theory to rewrite our grammars. This will enable existing plan recognition machinery to handle loops and plans with optional actions.

Using Context Free Grammars (CFGs) for plan grammars

It is uncontroversial that a limited form of Hierarchical Task Network plans(Ghallab, Nau, and Traverso 2004) can be represented as Context Free Grammars.(Erol, Hendler, and Nau 1994). As an example consider, the simple set of plans for traveling to a conference (GO2CONF) shown in Figure 1.

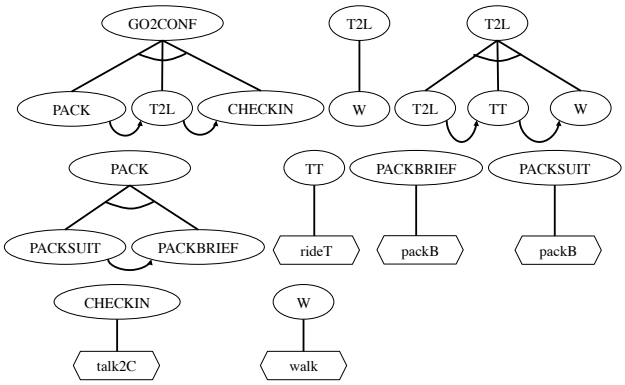

Figure 1: A simple set of Hierarchical Plans

These plans are encoded in the following CFG:

CFG: 1

$$GO2CONF \rightarrow PACK, T2L, CHECKIN$$
$$PACK \rightarrow PACKSUIT, PACKBRIEF$$
$$T2L \rightarrow W \mid T2L, TT, W$$
$$TT \rightarrow rideT$$
$$PACKBRIEF \rightarrow packb$$
$$PACKSUIT \rightarrow packs$$
$$CHECKIN \rightarrow talk2C$$
$$W \rightarrow walk$$

We will use a vertical bar to separate alternative rules that share a common left hand side (consider T2L). For brevity, we have shortened some of the longer names of the tasks to create non-terminals for the grammar. In this case: W is the non-terminal for walking, T2L is the non-terminal for the task of traveling to a location, TT is the non-terminal for traveling by train, PACK is the non-terminal that subsumes the two packing tasks of packing a suitcase (PACSUIT) and a briefcase (PACKBRIEF), and CHECKIN is the non-terminal for checking into the conference. For each of TT, PACKBRIEF, PACKSUIT, CHECKIN, and W there is a corresponding terminal that we assume to be observable in the world. We note all the actions here are totally ordered and all of the actions are required for the plan to be successful. We also note the grammar we have just given has a structure that exactly mirrors the structure of the HTN. While this is a very simple set of plans it will be sufficient for our discussion.

Once a set of plans has been encoded into a CFG the process of recognizing a plan is effectively producing the most likely parse of the observations(Geib, Goldman, and Maraist 2008; Geib 2009). In some situations we can even use the exact same set of algorithms that are used for natural language processing to produce such parses. However, typical parsing algorithms make a number of assumptions that make them unsuited to plan recognition. These assumptions include assuming the agent is only executing a single plan, all of the agent's actions are observable, that a complete execution of the plan is present, and others. As a result, researchers in plan recognition have often created their own algorithms for parsing plan grammars rather than using traditional natural language parsers.

YAPPR Background

Imagine that we take a CFG and for it we produce the set of left-most, depth-first parse trees. For the grammar in Figure 1 a subset of the trees that would result from this process is shown in Figure 2. Using tree substitution(Joshi and

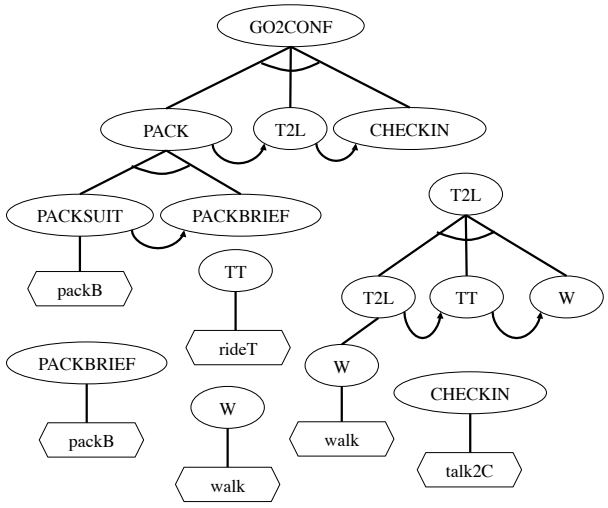

Figure 2: Leftmost derivation trees for a subset of the tasks shown in Figure 1

Schabes 1997), we can combine such trees to parse a set of

observations to produce larger plans for recognition. (Goldman, Geib, and Miller 1999; Geib and Goldman 2009).

The YAPPR plan recognition system embraces this kind of tree grammar based parsing but leverages a further fact. Since parses are extended by substitution, only the tree's frontiers need to be maintained. The action grammar's trees can be rewritten as a set of production rules, capturing just the required frontier non-terminals. Parsing can then be done as a rewriting process that is equivalent to tree substitution resulting in Plan Frontier Fragment Grammars (PFFG)(Geib, Goldman, and Maraist 2008).

Definition 1.1 *We define a* PFFG *as a tuple* $PL = \langle \Sigma, NT, P \rangle$ *where,* Σ *is a finite set of basic actions or terminal symbols,* NT *is a finite set of methods or non-terminal symbols, and* P *is a set of production rules of the form* $A, a \rightarrow \beta$ *where* $a \in \Sigma$*, and* β *is a string of symbols from* $(\Sigma \cup NT)^*$.

In the rest of this document non-terminal symbol names will be written in capitals and terminal symbol names in lower case. If a given PFFG production has an empty right hand side, we will denote the right hand side as a period. (Geib, Goldman, and Maraist 2008) provides a more complex definition of PFFGs that supports explicit representation of partial ordering within the grammar. For ease of exposition, in this definition we assume that all partial ordering has been addressed by explicitly enumerating all of the orderings in the grammar.

Using a PFFG, parsing is performed by considering the next observed terminal symbol. For any production for which the terminal symbol on the left hand side is the same as the next observation, and the non-terminal on the left-hand side is present in the current plan frontier, the non-terminal is removed from the plan frontier and replaced by the right hand side of the production. Effectively, the non-terminal on the left hand side can be rewritten as the string on the right hand side anytime the terminal on the left hand side is observed. In the case of rules with an empty right hand side, the non-terminal in question is removed and nothing is put into its place. With this grammar formalism in hand we return to our discussion of constructing the grammar given an initial CFG for the set of plans to be recognized.

In previous work (Geib and Goldman 2009), we have shown that to build the complete set of left-most, depth-first parse trees for arbitrary CFGs requires bounding the number of times that recursive productions in the original CFG can be called. Left recursive grammars (like our example grammar) produce infinite, descending, leftward parse trees by simply recursing. This is a valuable property. This is what allows our example grammar to recognize plans with an arbitrary number of train legs. The solution to this problem used in YAPPR is to bound the number of times the recursive productions can be used.

For example, assuming a limit of one recursive step, converting our original grammar to a YAPPR PFFG yields.

PFFG: 1

$$\begin{aligned} GO2CONF, packS &\rightarrow PACKBRIEF, T2L, CHECKIN \\ T2L, walk &\rightarrow . \mid TT, W \mid TT, W, TT, W \\ TT, rideT &\rightarrow . \\ PACKBRIEF, packB &\rightarrow . \\ CHECKIN, talk2C &\rightarrow . \\ W, walk &\rightarrow . \end{aligned}$$

Note the significant increase in the number of productions for T2L. Without the bound this would in fact have an infinite number of productions. Also note that we can only recognize plans that involve taking two trains or less. What about cases where we wanted to take three or even more? This would require setting a new, higher bound on the recursion depth and building a new PFFG.

As is well known in the computational theory and compilers literature(Hopcroft and Ullman 1979), parsing CFGs can be made easier by rewriting CFG grammars into normal forms. Such transformations generally have no effect on the grammar's expressiveness, but can have very significant impact on parsing efficiency. One such transformation that can be performed on a grammar is the removal of left recursion. Our previous work on YAPPR (Geib, Goldman, and Maraist 2008) limited the expressiveness of the grammars supported specifically to avoid cases of left recursion. Therefore we will look at using well-known rewriting methods from CFG theory to eliminate leftward recursion in the grammar and allow YAPPR to recognize fully recursive plans and therefore arbitrary looping.

Removing Left Recursion

Formal language theorists have show that left recursion can be removed from a CFG (Hopcroft and Ullman 1979; Wood 1987). This conversion is based on the insight 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, in general a production of the form: $A \rightarrow A\gamma \mid B$ can be rewritten as the two productions: $A \rightarrow B \mid BX; X \rightarrow \epsilon \mid \gamma X$. Based on this insight and following the algorithm provided in (Hopcroft and Ullman 1979) our original plan grammar (CFG: 1) can be converted to the following equivalent grammar:

CFG: 2

$$\begin{aligned} GO2CONF &\rightarrow PACK, T2L, CHECKIN \\ PACK &\rightarrow PACKSUIT, PACKBRIEF \\ T2L &\rightarrow W \mid WX \\ X &\rightarrow \epsilon \mid TT, W, X \\ TT &\rightarrow rideT \\ PACKBRIEF &\rightarrow packb \\ PACKSUIT &\rightarrow packs \\ CHECKIN &\rightarrow talk2C \\ W &\rightarrow walk \end{aligned}$$

that does not have left recursion. Note the introduction of the new non-terminal X that encodes the loop. This allows

the grammar to be written as right recursive rather than left recursive. We refer the interested reader to (Hopcroft and Ullman 1979) for more details on this algorithm.

Unfortunately, as we see in our example, the removal of left recursion can introduce ϵ-productions into the CFG. This is a problem, as the algorithm for generating PFFGs does not support ϵ-productions. Fortunately, the formal grammar theorists have also found an algorithm for removing ϵ-productions from a grammar.

The intuition behind the the ϵ-production removal is nothing more than multiplying out all the possible alternatives. The end result of removing them is that we effectively we include a production for each possible case (with and without the optional action). We again refer the interested reader to (Hopcroft and Ullman 1979) for details of this algorithm, but applying this simplification to following this transform results in the following CFG:

CFG: 3

$$\begin{aligned} GO2CONF &\rightarrow PACK, T2L, CHECKIN \\ PACK &\rightarrow PACKSUIT, PACKBRIEF \\ T2L &\rightarrow W \mid WX \\ X &\rightarrow TT, W \mid TT, W, X \\ TT &\rightarrow rideT \\ PACKBRIEF &\rightarrow packb \\ PACKSUIT &\rightarrow packs \\ CHECKIN &\rightarrow talk2C \\ W &\rightarrow walk \end{aligned}$$

The critical change to observe here is in the reformulation of the production for X.

It is worth noting that this transformation on a CFG can result in a loss of expressiveness. If the empty string was part of the language described by the original CFG it clearly cannot be part of the grammar with all ϵ-productions removed. Thus, the plan grammar can no longer recognize cases where the agent does nothing. While in formal terms this is a change does modify the expressiveness of the grammar it is not a significant loss for work done in plan recognition. It is rare that we want to recognize when an agent has performed no actions, and in those cases there are generally much simpler tests that we can perform than executing our plan recognition algorithms.

Having now removed both left recursion and ϵ-productions from the original CFG, we can convert our CFG:4 into a PFFG for recognition. This results in:

PFFG: 2

$$\begin{aligned} GO2CONF, packS &\rightarrow PACKBRIEF, T2L, CHECKIN \\ T2L, walk &\rightarrow . \mid X \\ X, rideT &\rightarrow W \mid W, X \\ TT, rideT &\rightarrow . \\ PACKBRIEF, packB &\rightarrow . \\ CHECKIN, talk2C &\rightarrow . \\ W, walk &\rightarrow . \end{aligned}$$

PFFG: 2 does not have any left recursion. This will allow us to use the existing YAPPR algorithm to recognize plans with arbitrary loops. Using the existing CFG simplification algorithms to first remove leftward recursion and then any ϵ-productions in the CFG of the plans to be recognized before creating a PFFG provides two benefits. First, it allows us to remove the need for a bound on the grammar in the creation of PFFGs. Second, it allows the original YAPPR algorithm to support recognition of plans with arbitrarily deep loops. We will return later to discuss the computational cost of this.

Optional actions

While we can think of optional actions within a plan as exceptionally short loops there is a critical difference. We notice in our initial CFG example that while we have a loop there are no ϵ-productions. Truly optional actions are either present or not. This requires ϵ-productions in the original grammar.

For example suppose the action of checking into the conference is actually optional. This would result in the following CFG:

CFG: 4

$$\begin{aligned} GO2CONF &\rightarrow PACK, T2L, CHECKIN \\ PACK &\rightarrow PACKSUIT, PACKBRIEF \\ T2L &\rightarrow W \mid T2L, TT, W \\ TT &\rightarrow rideT \\ PACKBRIEF &\rightarrow packb \\ PACKSUIT &\rightarrow packs \\ CHECKIN &\rightarrow \epsilon \mid talk2C \\ W &\rightarrow walk \end{aligned}$$

As we have already seen, there are existing algorithms for rewriting the CFG to removing such optional actions. Thus, if we invoke the process we have already outlined of removing left recursion and then removing all ϵ-productions we get the following CFG that is equivalent to CFG:4 :

CFG: 5

$$\begin{aligned} GO2CONF &\rightarrow PACK, T2L, CHECKIN \\ GO2CONF &\rightarrow PACK, T2L \\ PACK &\rightarrow PACKSUIT, PACKBRIEF \\ T2L &\rightarrow W \mid WX \\ X &\rightarrow TT, W \mid TT, W, X \\ TT &\rightarrow rideT \\ PACKBRIEF &\rightarrow packb \\ PACKSUIT &\rightarrow packs \\ CHECKIN &\rightarrow talk2C \\ W &\rightarrow walk \end{aligned}$$

and CFG:5 results in the following PFFG for recognition:

PFFG: 3

$$
\begin{aligned}
GO2CONF, packS &\rightarrow PACKBRIEF, T2L, CHECKIN \mid \\
&\quad PACKBRIEF, T2L \\
T2L, walk &\rightarrow . \mid X \\
X, rideT &\rightarrow W \mid W, X \\
TT, rideT &\rightarrow . \\
PACKBRIEF, packB &\rightarrow . \\
CHECKIN, talk2C &\rightarrow . \\
W, walk &\rightarrow .
\end{aligned}
$$

Thus we have identified algorithmic methods for modifying CFG grammars that allow YAPPR to handle loops and optional actions.

Relation to GNF

The YAPPR compilation algorithm results in PFFGs that are similar to Greibach Normal Form (GNF) (Greibach 1965)

Definition 1.2 *We define a CFG as being in* ***Greibach Normal Form****(GNF) if every production of the grammar has the form* $A \rightarrow a\beta$ *where* A *is a non-terminal,* a *is a terminal and* β *is a possibly empty string of non-terminals.*

In fact, to convert a CFG to GNF, the two transformations we have discussed must be performed first. We note that PFFGs are not as restrictive as GNF, since a PFFG can have multiple terminals in the right hand side of a rule. Further, GNFs are often formalized as limiting the right hand side of a production to a length of three elements (one terminal and not more than two non-terminals) which we do not do. While this does not change the expressivity of the grammar in general it would significantly impact the use of the grammars a PFFGs as it would bound the trees to have at most two non-terminals on their frontier.

The Probability Model

We have claimed that no changes are needed to the YAPPR algorithm in order to recognize plans with looping and optional actions. However while the parsing algorithm used by YAPPR does not need modification, the probability model used by YAPPR does need modification to correctly capture the probability of these control structures. YAPPR's probability model is given by the following formula:

Definition 1.3

$$
Pr(exp \wedge obs) = \prod_{i=0}^{I} Pr(G_i) \prod_{i=1}^{n} (Pr(rule_i) \prod_{k=1}^{n} (1/|ext(AP(exp_k)|)
$$

where there are n observations.

The first term captures the likelihood of each of the root goals present in the parse, and needs no modification to handle loops and optional actions. The second term captures the likelihood of each of the given productions being used to account for the observed action. This principally captures the structural assumptions that underlie the particular PFFG production. The third term captures the likelihood that of the possible actions, that are enabled by the agent's previous actions, the agent chose this particular action next. Both the second and third terms have previously been modeled by a uniform distributions across all the possible options (all of the alternatives were considered equally likely). This assumption is *not* forced by the model, though. Non-uniform distributions *could* be used.

Implicitly both the second and third terms of this formula include independence assumptions violated by the presence of looping and optional actions. In the case of the second term, we would be assuming that all of the possible permutations of the optional actions are equally likely and have the same likelihood as any other actions that could be done next. In the case of the third term we would be assuming that at any given time step the loop was as likely to terminate as to continue, and again both of these options would be considered as likely as any other action the agent could perform. Neither of these assumptions are likely to hold in general.

Conveniently, it is relatively easy to change the probability models used by YAPPR. The use of looping and optional actions strongly argue for using more complex probability models for the second and third term. Such models should be conditioned on structural features of the existing plan and possibly even features of the current world state. The correct form for these new models is an open problem and area of active research.

Impact

We note that both the transformation for removing looping, the removal of ϵ-productions from the grammar increase in the size of the resulting grammars. This can have very significant impact on the run-time of the parsing. Thus, while we have shown the very nice theoretical result that the YAPPR algorithm can support looping and optional actions without changes to the parsing algorithm there is still a significant question of the practicality of doing so.

As pointed out in (Geib 2004) the chief cost of plan recognition for systems like YAPPR is in the number of possible parses that must be maintained, and there are a few generalizations that we can draw about the impact of loops and optional actions on the number of parses that YAPPR will have to consider.

Relative to the number of parses, lopping inflicts a relatively low cost. YAPPR will be required when in the loop to maintain two explanations, one for the possibility that the loop will terminate on the next step and one for the possibility that the loop will continue. Thus the number of explanations will be doubled. While this can be a significant cost it is often dwarfed by the costs of optional actions.

As we have seen, dealing with optional actions amounts to building a new production for each of the possible sequences of actions that could result from the presence or absence of the optional actions. For multiple optional actions this results in an exponential number of productions, and hence an exponential increase in the number of parses that must be considered when parsing a task that contains optional actions. Thus, depending on the number and location of the optional actions, optional actions can actually have far more significant impact on the run-time of the YAPPR algorithm.

While we have shown that YAPPR can be used to recognize plans with loops and optional actions, a significant amount of empirical evidence should be gathered before concluding that recognizing plans with these structures can be be done efficiently enough for any particular application.

Conclusions

Viewing plan recognition as a parsing task allows one to leverage known results from work on formal grammar parsing. In this paper we have shown specifically how this can be done for one particular grammar based plan recognition algorithm. We have shown its relation to existing formal grammar work and how it can be used to increase the expressiveness of plan recognition algorithms. While we have discussed these results in the context of a single plan recognition system, the formal ideas behind it are sufficiently general to apply to any plan recognition system that is based on CFGs and parsing of observations.

Acknowledgements

The authors would like to thank John Marist for all his work on the YAPPR system. Dr. Geib's work in this paper was supported by DARPA/U.S. Air Force under Contracts FA8650-06- C-7606 and the EU Cognitive Systems project PACO-PLUS (FP6- 2004-IST-4-027657) funded by the European Commission.

References

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2005. Fast and complete symbolic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

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. MIT Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task network planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS 94)*, 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 of the International Conference on Automated Planning and Scheduling (ICAPS)*, 81–89.

Geib, C. 2004. Assessing the complexity of plan recognition. In *Proceedings of AAAI-2004*, 507–512.

Geib, C. 2006. Plan recognition. In Kott, A., and McEneaney, W., eds., *Adversarial Reasoning*. Chapman and Hall/CRC. 77–100.

Geib, C. W. 2009. Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1702–1707.

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

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A new model of plan recognition. In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*.

Greibach, S. 1965. A new normal form theorem for context-free phrase structure grammars". *Journal of the ACM* 12(1):42–52.

Hoogs, A., and Perera, A. A. 2008. Video activity recognition in the real world. In *Proceedings of the Conference of the American Association of Artifical Intelligence (2008)*, 1551–1554.

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

Huber, M., and Simpson, R. 2004. Recognizing the plans of screen reader users. In *Proceeding of the AAMAS Workshop on Modeling Others from Observations (MOO)*.

Joshi, A., and Schabes, Y. 1997. Tree-adjoining grammars. In *Handbook of Formal Languages, Vol. 3*, 69–124. Springer Verlag.

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 of the 2000 Conference on Uncertainty in Artificial Intelligence*, 507–514.

Vail, D. L., and Veloso, M. M. 2008. Feature selection for activity recognition in multi-robot domains. In *Proceedings of the Conference of the American Association of Artifical Intelligence (2008)*, 1415–1420.

Wood, D. 1987. *Theory of Computation*. 10 East 53rd Street, New York, NY 10022: Harper and Row.