Project no.:	**027657**
Project full title:	**Perception, Action & Cognition through learning of Object-Action Complexes**
Project Acronym:	**PACO-PLUS**
Deliverable no.:	**D4.3.4**
Title of the deliverable:	**Technical Report: Emperical Evaluation of Lexicalized Plan Recognition**

Contractual Date of Delivery to the CEC:	31 January 2010	
Actual Date of Delivery to the CEC:	19 February 2009	
Organisation name of lead contractor for this deliverable:	UEDIN	
Author(s): Christopher Geib, Ronald Petrick, and Mark Steedman		
Participant(s): UEDIN		
Work package contributing to the deliverable:	WP4	
Nature:	R	
Version:	Final	
Total number of pages:	14	
Start date of project:	1st Feb. 2006	**Duration:** 48 month

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)
Dissemination Level

PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

A probabilistic plan recognition system called ELEXIR (Engine for LEXicalized Intent Recognition) has been developed on the Paco-Plus project. ELEXIR explores the possiblity of using existing lexicalized gramatical formalisms of the kinds used in WP5 and can be mapped to OACs to represent plans (of the kind seen in WP4.1 and WP4.2) that are to be recognized by the system. Given this focus on lexicalized grammars, ELEXIR makes heavy use of the idea of "headedness" and takes this idea from the NL literature for use in the plan recognition literature. This deliverable focuses on describing the ELEXIR system and its formalization as well as experiments on its scalability. Specifically it focuses on the use of headedness decisions in plan representation to control the search space of possible plans.

Keyword list: Probabilistic Plan Recogntion, Symbolic Plan Recognition, Lexicalized Plan Grammars, Experimental validation

Table of Contents

1. Executive Summary

Plan recognition(PR) is the inference of an agent's high level plans and goals based on observations of their actions. We distinguish PR from other work reported in WP3 on activity recognition. PR focuses on the recognition of sequences of high level actions as plans for high level goals. The kind of activity recognition reported in WP3 should be viewed as a necessary precursor to the recognition task we discuss here. For example, activity recognition for recognizing grasping actions would be one part of a process that would collect low level sensor inputs, and form them into repeatable "basic actions". A sequence of such basic actions would constitute the observation stream taken as input to the kind of PR algorithms discussed here.

PR is necessary for PACO-PLUS style intelligent agents in order to recognize high level plans of other agents. For example, in order for an agent to recognizing a sequence of grasping and moving actions as the robot cleaning up the kitchen, the basic grasping, moving, and releasing actions must all be combined into a higher level plan for the cleaning task.

The two documents included in this deliverable report on a number of significant research developments in PR:

- Formalization of how plans can be represented in a lexicalized grammar, specifically Combinatory Categorial Grammars (CCGs)[1]. (Appendix A)
- Formalization of the idea of headedness and plan heads for use in plan recognition. (Appendix A)
- Experimental evidence of the value of headedness in unambiguous environments. (Appendix A)
- Distinguishing three different possible sources of ambiguity in plan recognition called *action*, *syntactic*, and *attachment* ambiguity. (Appendix B)
- Experimental evidence of the value of headedness in domains with high syntactic ambiguity. (Appendix B)

While this work is a significant research contribution in its own right, it should be seen as fitting clearly into the PACO-PLUS research program. As such the PR work described in this deliverable is heavily influenced by the research objectives of PACO-PLUS. While, the work reported here is not explicitly formulated in terms of PACO-PLUS' Object Action Complexes(OACs) the representations used here are consistent with this approach. As such this work echos a number of themes taken from the PACO-PLUS project:

- The understanding of CCG categories as functional types fits well with the formulation in WP4 of OACs as functions.
- OACs are grounded in observable embodied actions, similarly in lexicalized grammars all syntactic categories are tied to an observable action.
- grounding of actions and lexicalized representations of action come together in a concept we call "headedness" of plans taken from natural language.
- headedness is a new idea in the representation of plans and plan recognition and has a significant impact on the runtime of the plan recognition algorithm. This strengthens the PACO-PLUS contention that efficient intelligent systems much be grounded in real world experience.

Thus, the work included in this deliverable should be seen both as a strong contribution to plan recognition research and as a contribution to the PACO-PLUS research agenda.

2. Papers Included in D4.3.4

[A] **Delaying Commitment in Plan Recognition Using Combinatory Categorial Grammars**
Christopher Geib
Published in the proceedings of the International Joint Conference on Artificial Intelligence 2009 (IJCAI-09).

Abstract: This paper presents a new algorithm for plan recognition called ELEXIR (Engine for LEXicalized Intent Recognition). ELEXIR represents the plans to be recognized with a grammatical formalism called Combinatory Categorial Grammar(CCG).We show that representing plans with CCGs can allow us to prevent early commitment to plan goals and thereby reduce runtime

[B] **Lexical Ambiguity and its Impact on Plan Recognition**
Christopher Geib
In submission to the International Conference on Automated Planning and Scheduling 2010 (ICAPS-10).

Abstract: Viewing plan recognition (PR) as a parsing problem, this paper distinguishes three sources of ambiguity: *action ambiguity*, *syntactic ambiguity* and *attachment ambiguity*. Previous work PR in has often conflated these different sources of ambiguity. This paper clarifies this distinction and explicitly studies the effect of syntactic ambiguity on the runtime of a particular PR algorithm. It also argues for new method for controlling plan level ambiguity in probabilistic PR based on the idea of plan "heads" using lexicalized grammars.

References

[1] Mark Steedman. *The syntactic process*. MIT Press, 2001.

Appendix A

Delaying Commitment in Plan Recognition Using Combinatory Categorial Grammars

Christopher W. Geib

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

Abstract

This paper presents a new algorithm for plan recognition called ELEXIR (Engine for LEXicalized Intent Recognition). ELEXIR represents the plans to be recognized with a grammatical formalism called Combinatory Categorial Grammar(CCG). We show that representing plans with CCGs can allow us to prevent early commitment to plan goals and thereby reduce runtime.

1 Introduction

Given a plan library and a set of observations, the problem of identifying an agent's plans and goals on the basis of their observed actions is called *plan recognition* (PR), and is a well studied problem in AI. Much of the prior research on PR [Bui *et al.*, 2002; Avrahami-Zilberbrand and Kaminka, 2005; Geib, 2006; Kautz, 1991] use algorithms that make early commitments to hypothesized root goals and sub-plans. This creates a problem. As [Geib, 2004] has pointed out, such early commitment can result in maintaining an exponential number of hypotheses. Many, of these hypotheses will be discarded later as being impossible. Thus, early commitment to hypotheses can needlessly increase runtime.

To address this problem, we will formulate PR based on Combinatory Categorial Grammars (CCGs)[Steedman, 2000], a grammatical formalism developed for use in natural language parsing(NLP). Using CCGs to represent plan libraries will require us to introduce the new idea of *plan heads*. We will show that making the correct choices about plan heads enables a least commitment approach to plan recognition and reduces runtimes.

In the rest of this paper, we will outline our approach to plan recognition. We then show how to represent plans in CCGs and define plan heads. We will then present a new, probabilistic plan recognition algorithm called ELEXIR(Engine for LEXicalized Intent Recognition) based on these ideas. We will discuss its theoretical complexity, and an empirical evaluation of its performance. These experiments will show that correct choices for plan heads enable significant computational saving.

We note, the relationship between PR and NLP is not a new idea, and there is previous work in using ideas from NLP in PR including [Carberry, 1990; Pynadath and Wellman, 2000] and others. However, we know of no prior work using CCGs and headedness to control early commitment.

2 Intuitions and an Example

We are interested in *probabilistic* plan recognition, and will use weighted model counting to solve it. We assume as given a set of *observations* and a *CCG* specification of a *plan lexicon* defining the plans to be recognized. To perform PR, we advocate parsing the observations into the complete and covering set of *explanations* that organize the observations into one or more plan structures meeting the requirements defined in the plan lexicon. We then establish a probability distribution over the explanations to reason about the most likely goals and plans. To do this, we must encode the plans in CCGs. An example will help show how to do this.

Consider the simple abstract hierarchical plan drawn as a partially ordered AND-TREE shown in Figure 1. To execute

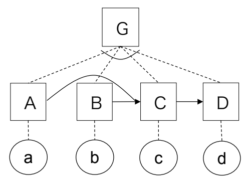

Figure 1: An abstract plan with partial order causal structure

action **G** the agent must perform actions **A**, **B**, **C**, and **D**. **A** and **B** must be executed before **C** but are unordered with respect to each other, and finally **D** must be performed after **C**.

3 Representing Plans in CCG

To represent the example plan in a CCG, each observable action is associated with a set of *categories*.

Definition 3.1 *We define a set of categories, C, recursively:*

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

Complex categories *: If* $Z \in C$ *and* $\{W, X, ...\} \neq \emptyset \subset C$*, then* $Z\backslash\{W, X, ...\} \in C$ *and* $Z/\{W, X, ...\} \in C$.

Intuitively, complex categories can be thought of as functor categories that can take a set of *arguments* ($\{W, X, ...\}$) and produce a *result* (Z). The direction of the slash indicates where the functor 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.

Thus, an action with the category $A\backslash\{B\}$ is a function that results in performing action A in contexts where an action with category B has already been performed. Likewise $A/\{B\}$ is a function that results in performing A if an action with category B is executed later.

We are now in a position to define a plan lexicon.

Definition 3.2 *We define a **plan lexicon** as a tuple* $PL = \langle \Sigma, C, f \rangle$ *where,* Σ *is a finite set of observable action types,* C *is a set of possible CCG categories, and* f *is a function such that* $\forall \sigma \in \Sigma, f(\sigma) \rightarrow C_\sigma \subseteq C$.

C_σ is the set of categories an observation of type σ can be assigned. As a short hand, we will often provide just the function that maps observable action types to categories to define a plan lexicon. For example,

$$a := A, \quad b := B, \quad c := (G/\{D\})\backslash\{A, B\}, \quad d := D.$$

defines one plan lexicon for our example plan. The following definitions will also be helpful:

Definition 3.3 *We define a category R as being the **root** or **root-result** of a category G if it is the leftmost atomic result category in G. For a category C we denote this root(C)*

Thus G is the root-result of $(G/\{D\})\backslash\{A, B\}$. Further,

Definition 3.4 *we say that observable action type* a *is a possible **head** of a plan for C just in the case that the lexicon assigns to* a *at least one category whose root-result is C.*

In our lexicon c is the head for G.

This formulation of CCGs is closely related that of [Baldridge, 2002] in allowing sets of arguments to categories. Sets of arguments are critical for our treatment of partial ordering in the plan. For example, the first argument to c's category is the leftward looking set $\{A, B\}$ representing the partial ordering of these actions before C. This definition also allows multiple categories to be associated with an observed action type. However, for ease of exposition, we will suppress notation for this if an observation only has a single category.

Next we must show how CCG categories are combined into higher level plan structures. In CCGs *combinators* [Curry, 1977] are used to combine the categories of the individual observations. We will only use three combinators defined on pairs of categories:

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

where X and Y are categories, and α and β are possibly empty sets of categories. Other Combinatory rules are sometimes used in NLP[Steedman, 2000], however, we leave the use of these combinators in the PR context for future work.

To see how a lexicon and combinators parse observations into high level plans, consider the derivation in Figure 2 that parses the sequence of observations: a, b, c.

$$\begin{array}{ccc} a & b & c \\ \overline{A} & \overline{B} & \overline{(G/\{D\})\backslash\{A, B\}} \\ & \multicolumn{2}{c}{\overline{(G/\{D\})\backslash\{A\}}^{<}} \\ \multicolumn{3}{c}{\overline{G/\{D\}}^{<}} \end{array}$$

Figure 2: Parsing Observations with CCGs

As each observation is encountered, it is assigned a category on the basis of the lexicon. Combinators then are used to combine the categories. First, a is observed and assigned A and no combinators can be applied. Next we observe b, and it is assigned B. Again, none of the combinators can be applied. Notice however, all the hierarchical structure from the original plan for achieving G is included in c's category. Therefore, once c is observed and assigned its category, we can use leftward application twice to combine both the A and B categories with c's initial category to produce $G/\{D\}$.

3.1 Designing Plan Lexicons

In the preceding discussion, we have avoided some of the representational questions in designing a plan lexicon. The critical choice made during lexicon construction is which action types will be the plan heads. Different choices for heads result in different lexicons. For example, the following is an alternative lexicon for G where d is the head rather than c.

$$a := A, \quad b := B, \quad c := C, \quad d := (G\backslash\{A, B\})\backslash\{C\}.$$

We can also represent the plan for G with the following lexicon where a has two possible head categories for G:

$$\begin{array}{l} a := \{\ ((G/\{D\})/\{C\})/\{B\}, \\ \qquad ((G/\{D\})/\{C\})\backslash\{B\}\ \}, \\ b := B,\ c := C,\ d := D. \end{array}$$

There are also a number of still more complex lexicons where other choices are made for the heads.

Modeling issues that are similar to choosing heads for CCGs occur in traditional hierarchical task network (HTN) representations[Ghallab *et al.*, 2004] in the form of choosing the sub-goal decomposition. With their long tradition in planning, decisions about what is and isn't a sub-goal in a single level of an HTN may seem quite intuitive. However, like choosing heads for a CCG this is a design decision for HTNs and can have serious impact on PR and planning algorithms. We will say more about how to choose CCG heads later in this paper.

Keep in mind, we want to use parsing of CCGs to build explanations for the observed actions. However, we don't want to make early commitments to goals. In contrast to traditional HTNs, CCG categories function as a tree and/or subtree spine crossing multiple levels of plan decomposition. We can use the "vertical slicing" of plans by categories to define the scope of our commitments in building goal and plan hypotheses. We state the following principle:

Principle of minimal lexically justified explanation: In building explanations we never hypothesize any plan structure beyond that provided by the categories of the observed actions in the plan lexicon.

This principle clearly defines when, how much, and what kind of plan structures and hypothesis we can build. It enables a least commitment approach in that it limits plan hypothesis to those for which we have observed the head of the plan. The choice of heads for plans will now allow us to determine when commitments are made about goals, sub-goals, and plans. As we will see next, it also enables a simple algorithm for generating explanations for observations.

4 Building Explanations in ELEXIR

While we would like to use NLP parsing algorithms for explanation construction, there are differences between these problems that prevent this. In the case of PR, we can't bound a-priori how many observations there will be. Further, we can't assume that all of the observations must contribute to a single goal. We can't even assume that we have seen all of the observations associated with the plan. Many well known parsing algorithms like CKY, even when modified for CCGs [Steedman, 2000], leverage some or all of these assumptions and are therefore unusable. Therefore we must provide our own algorithm for parsing action categories into explanations.

For ease of computation we will restrict our action grammars to only *leftward applicable* categories.

Definition 4.1 *We define a set of categories C^L as* ***leftward applicable*** *if and only if*

1. $C^L = C^A \cup C^C$ *and*
2. C^A *is a set of atomic categories and*
3. C^C *is a set of complex categories of the form* $X\{/Y_i\}^*\{\backslash Z_j\}^*$ *such that* $X \in C^A$ *and* $\forall i, Y_i \subseteq C^A$ *and* $\forall j, Z_j \subseteq C^A$.

Intuitively all of the leftward looking arguments in a category must precede (be "outside") all of the rightward looking arguments. Thus $(((A/\{B\})/\{C\})\backslash\{D\})\backslash\{E\}$ is a leftward applicable category but$(((A/\{B\})\backslash\{C\})/\{D\})/\{E\}$ is not. We will return shortly to discuss the reasons for this limitation.

Definition 4.2 *We next define an* ***explanation*** *for a sequence of observation instances for each time instance $\sigma_{t1}...\sigma_{tn}$ given a plan lexicon $PL = \langle \Sigma, C^L, f \rangle$ as a sequence of categories $[c_1...c_i]$ that result from parsing the input stream on the basis of the plan lexicon.*

We can now provide a simple algorithm to generate all the explanations for a set of observations. See Figure 3. The intuition for the algorithm is as follows. For each explanation and for each category that the current observation could be assigned, check that all of its leftward looking arguments are present in the current explanation. If so, we clone the current explanation, add the category to the explanation, and use application to remove all of its leftward looking arguments. Then for each category in the explanation that could combine with the new category using rightward composition or application, duplicate the explanation and execute the composition in the new copy. Add the new explanation to the set of explanations and repeat for the next observation.

To remain consistent with the plan lexicon, the algorithm cannot assign a category to an observation unless all of the category's leftward arguments have been observed. To do so would hypothesize explanations that violate the ordering constraints specified in the plan lexicon. Restricting our grammars to leftward applicable categories simplifies this test, captured in the IF clause at the center of the algorithm.

```
Procedure BuildExplanations(σ_t1...σ_tn) {
 ES = { [ ] };
 FOR i = 1 to n
     ES' = ∅;
     FOR each exp = [c_1...c_j] ∈ ES
         FOR each c ∈ f(σ_ti);
             IF all of c leftward arguments are in exp, and can
                     be removed from exp in order, THEN
                 LET [c_1...c_k] be exp with all of c's leftward
                   arguments removed by function application
                   and c' be the result of c with its leftward
                   arguments removed.
                 ES' = ES' ∪ [c_1...c_k, c]
                 FOR each c_m ∈ [c_1...c_k] such that there exists
                         a combinator that will compose c_m
                         and c' resulting in c".
                     exp' = remove(c_m, [c_1...c_k, c]
                     ES' = append(exp', c").
                 END-for;
             END-if;
         END-for;
     END-for;
     ES = ES';
 END-for;
return ES; }
```

Figure 3: High level algorithm for explanation generation.

Thus, the algorithm incrementally creates the set of all explanations by assigning categories, discharging leftward looking arguments, and then applying each possible rightward looking combinator between the existing categories and the categories introduced by the current observation.

For example, given the original lexicon and the observations: a, b, c, d the algorithm produces $[G]$ and $[G / \{D\}, D]$ as the explanations. Note, the second explanation is included to account for the case where the D category will be used in some other, as yet unseen, plan. Under the assumption that a given category can only contribute to a single plan, if these categories are consumed at the earliest opportunity they will be unavailable for later use. Since all leftward arguments are discharged when assigning an observation a category, and each possible combinator is applied as later categories are added, this algorithm is complete and will produce all of possible explanations for the observations.

5 Computing Probabilities in ELEXIR

The above algorithm computes the exclusive and exhaustive set of explanations. Given this, if we can compute the conditional probability of each explanation, then the conditional probability for any particular goal is just the sum of the probability mass associated with those explanations that contain it. More formally:

Definition 5.1

$$P(goal|obs) = \sum_{\{exp_i|goal \in exp_i\}} P(exp_i|obs)$$

where $P(exp_i|obs)$ is the conditional probability of explanation exp_i. Therefore, we need to define how to compute the conditional probability for an explanation.

There are a number of different probability models used to compute the probability of a CCG parse in the NLP literature [Hockenmaier, 2003; Clark and Curran, 2004]. We will extend one described in [Hockenmaier, 2003]. For an explanation, exp, of a sequence of observations, $\sigma_1...\sigma_n$, that results in m categories, $c_1, ..., c_m$, in the explanation, we define the probability of the explanation as:

Definition 5.2

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

Where $cinit_i$ represents the category initially assigned in this explanation to observation σ_i. Thus, the first product represents the probability of each observation having their assigned initial CCG categories. This is standard in NLP and assumes the availability of a probability distribution over the observation's set of categories.

The second term captures the probability that each category will not be combined into a larger plan but itself represents a separate plan. This is not part of traditional NLP models. In NLP it makes no sense to consider the probability of multiple interleaved sentences or fragments. However, this assumption does not hold for PR. It is more than possible for a given sequence of observations to contain multiple interleaved plans or to only cover fragments of multiple plans being executed (consider multi-day plans). Therefore, our system must be given a prior probability for each category that occurs as a root-result in the lexicon. The role of these priors in Definition 5.2 requires some discussion.

We will denote the multiset of all values of $root(c_j)$ for a given explanation, as exp_{Goals}, and the probability of this particular multiset of root-result categories being adopted as top-level goals as $P(exp_{Goals})$. Keep in mind, in ELEXIR we want to allow for multiple instances of a given result in exp_{Goals} (it is acceptable for $root(c_i) = root(c_j)$ where $i \neq j$).

We denote the set of categories in exp_{Goals} as $Goals$. Finally, we represent the assumed probability of an agent adopting a particular root-result c as a goal as $P(c)$ with each instance of c in exp_{goals} being chosen (or rejected) independently. This means the probability that there will be exactly n instances of category c in exp_{Goals} is given by $P(c)^n(1 - P(c))$.

This is almost certainly incorrect – intuitively the probability of multiple instances of a single goal decreases far more rapidly than this, making this an over estimate of the likelihood of the goals. The algorithm supports more sophisticated probability models, and this is an area for future work.

If we let $|Goals_c|$ represent the number of instances of category c in exp_{Goals}:

$$P(exp_{Goals}) = \prod_{c \in Goals} P(c)^{|Goals_c|}(1 - P(c)) \prod_{c \notin Goals} (1 - P(c)).$$

Collecting all of the $1 - P(c)$ terms produces a product over all the categories in the lexicon and is therefore a constant:

$$P(exp_{Goals}) = \prod_{c \in Goals} P(c)^{|Goals_c|} K$$

Rewriting in terms of the instances in the explanation yields the second term seen in Definition 5.2.

$$P(exp_{Goals}) = \prod_{j=1}^{m} P(root(c_j)K$$

6 Complexity Analysis of ELEXIR

Having completed the description of the algorithm and probability model, we briefly consider its theoretical complexity. In order not to be distracted by the number of possible explanations computed, we consider how efficient the algorithm is in computing a single explanation for n observations.

We begin by noting that testing for the equivalence of two categories (and hence for combinator applicability) for any particular CCG is a constant time operation. Since each category can be though of as a tree, testing equality is equivalent to doing an in-order traversal. However, since the CCG grammar is fixed, we know the size of the largest category, and can then treat this cost as a constant,C.

The algorithm has two stages, explanation building and computing probabilities. We discuss each separately.

Explanation Building 1) **Discharging leftward arguments**: Let K be the fixed size of the grammar's largest leftward looking argument set. Verifying that all K arguments have been seen costs CK operations for each of the possibly $n - 1$ previous categories. This results in a worst case $O(n)$ cost.

2) **Applying combinators**: Let J be the fixed number of combinators. The algorithm must test each new category against each of the (in the worst case) $n - 1$ preceding categories. This results in nCJ tests for each observation for an $O(n)$ cost.

Computing Probability Computing the first term of the probability can be done in constant time when the category is chosen. The second term requires a single multiplication for each of the categories in the explanation. The cost of this is bounded above by $O(n)$.

Thus the worst case complexity for building a single explanation is $O(n)$. We also note this is as efficient as any algorithm can be since each of the observations has to be considered. Therefore the effective runtime of ELEXIR hinges most critically on the number of explanations being built. We argue that a least commitment approach can control the number of explanations being built by correctly choosing plan heads. We will examine this claim in the next section.

7 Empirical Analysis of ELEXIR

To verify the correctness of our system and to test our hypothesis about the efficacy of headedness we have developed a testing harness that allows us to systematically vary a number of parameters that define the plans in the CCG plan lexicon. These parameters include:

- **order**: How many and what type of ordering constraints exist between the actions in the plans. This parameter can take on the following values:
  - *Total*: actions in a sub-plan are totally ordered.
  - *First*: each sub-plan has a designated first action. All other actions in the plan are ordered after it but are unordered with respect to each other.
  - *Last*: each sub-plan has a designated last action. All other actions in the sub-plan are ordered before it, but are unordered with respect to each other.
  - *Unord*: actions in a sub-plan are unordered.
- **depth**: The depth of each plan.
- **num-roots**: The number of plans in the lexicon.
- **and-bf**: The number of children for each sub-plan.
- **headedness**: Determines which sub-plan step will be the head. This ranges between 0.0 (leftmost/"first") and 1.0 (rightmost/"last").

To create these plans, **num-roots** complete hierarchical plans based on AND-trees obeying **depth** and **and-bf** were generated and ordering constraints were established over each sub-tree. These plans were then converted to a CCG lexicon by starting at the root of the plan and recursively descending the tree following the actions with the indices given by ⌈ (**headedness** * **and-bf**) ⌉ collecting siblings that are to the left and the right of the action. When a leaf is reached a CCG category is built maintaining the ordering constraints of the original plan. This process is repeated for all sub-plans not covered by the initial category.

Given a CCG plan library we generated observations to test the system by randomly selecting a root-result category and producing a plan instance for it based on the plan library. (For test cases with multiple plans this process was repeated and the resulting plan instances were interleaved, maintaining the ordering constraints in the individual plans.) ELEXIR is then timed computing the conditional probability of all the root-results found by the algorithm given CCG plan library and the sequence of observations.

All of our experiments on our Allegro Common LISP 8.1 implementation of ELEXIR were conducted on a MacBook with 4Gb of main memory and 2 2.2-GHz CPUs. We report CPU time exclusive of any time used by garbage collection, the operating system or by other processes. For cases where the runtime registered as zero we report a runtime of 1 msec.

As a first exploratory test of the system we set **roots** to twenty, **and-bf** to three, and **depth** to two. We then ran a full factorial experiment on all values of the **order** factor and **headedness** at values of 0.001, 0.5, and 1.0. Each data-point had two interleaved plans resulting in a total of eighteen observations. ELEXIR achieved one hundred percent accuracy on this input data recognizing both plans in the input stream with the majority of the runs completing in under a second. These results verify the correctness of our implementation and its accuracy in the case of no noise or ambiguity.

7.1 Reducing Runtimes by Choosing Plan Heads

The central claim of this paper is that using CCGs and the correct choice of plan heads can delay commitment to plan and goal hypothesis and thereby reduce runtimes for PR systems. To validate these claims, we need to compare the system's runtimes varying the headedness of the plans. Synthetic data provide the perfect means for us to vary headedness of plans while controlling for other variables.

Notice that previous work in PR that make early commitments to plans and goals are effectively always operating with plans libraries that have a **headedness** value fixed at zero. If we fix **headedness** at zero, then each category is effectively a left most depth first tree with no leftward arguments. Thus when the first action of a plan is seen the whole left spine of the tree is introduced with the category, and all subsequent observations are also left most depth first trees. Thus, **headiness** values very close to zero make the same early commitment that we argued against in other PR systems.

This means we can use very low **headedness** values as the baseline for our experiments. If we see a drop in runtime as **headedness** is increased, this confirms our hypothesis that moving the head later in the plan delays commitments to the goal hypothesis and reduces the algorithm's runtime.

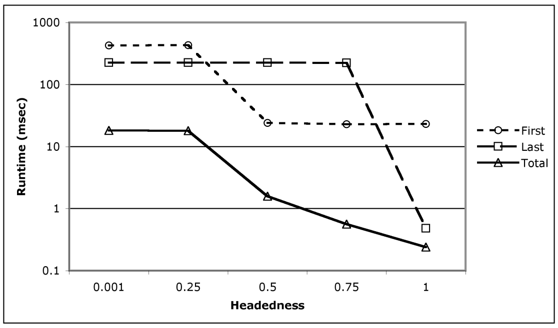

Figure 4: Average Runtimes for Order First, Last, and Total Plans. Each point represents the average of 500 test runs.

Figure 4 displays the results for a full factorial experiment where each test case was taken from a plan lexicon with **num-roots** set to one hundred, and each plan had an **andbf** of four. The tested factors were **order** and **headedness**, and they varied between *total*, *first*,*last* and 0.001, 0.25, 0.5, 0.75, 1.0 respectively. All other factors were held constant at their previous values. By setting **headedness** to these values each of the children of each AND-node is, in turn, treated as the head of the plan. The steady drop in runtime across all values of **order** as the head of the plan is moved to the right provides very convincing evidence for our claims.

We see a significant decrease in runtime for all ordering cases as the head is moved later in the plan and commitment to plan structure is delayed. We note all of the gains for the **order** *first* case are almost immediate while the gains for the *last* case do not occur until much later. Considering the ordering constraints in the respective plans will explain this.

In the **order** *last* case, we do not see improvement in the runtime until the head of the plan is assigned to the last action. In this case, since all the leading actions are unordered with respect to each other, any commitment to the structure of the plan before the last action is equivalent in runtime, but delaying commitment to the plan structure until the final action results in significant savings.

In the case of the **order** *first*, a value of 0.001 for headedness aligns the head of the plan with the causally first action of the plan. As we move the head later in the plan we get an initial drop in runtime as one of the unordered actions is selected, but no significant later savings since the ambiguity associated with the unorded actions is being moved from one side to the other of the head action.

We did not identify **headedness** as having a significant effect in completely unordered plans. The lack of structure in these plans means that whenever an action in one of these plans is observed ELEXIR is required to consider an exceptionally large number of hypotheses, but moving the head does not restrict the number of hypotheses. This should not be seen as a significant limitation. We believe completely unordered plans are unlikely in the real world.

7.2 Discussion and Limitations

These experiments show that a PR algorithm based on CCGs and headedness is viable and provides a principled way to control early commitment. However, we have not provided an answer for how to choose plan heads during lexicon design. These decisions have to be made by considering three key factors:

1. Criticality of early recognition: In cases where early recognition is critical, choosing a head that is early in the plan is better. Earlier heads allow earlier recognition and must be weighed against the runtime. We can certainly imagine domains where the need for early recognition outweighs the runtime costs.
2. Runtime: In general, as we have shown, to minimize runtime, choosing actions that fall later in the plan as heads is better.
3. Causal structure: We can see in these experiments aligning choices of plan heads with the causal structure produces the greatest computational wins.

Thus, all three of these features must be considered by the system builder when encoding a PR domain.

It is worth noting that the algorithm given here does have a significant limitation. It is unable to compute the probability for any plan for which the head has not been observed. Consider the first example CCG lexicon given for the initial example. Suppose the system is only given two observations $[a, b]$. Intuitively this should give us a significant amount of evidence for the goal G. However, the category with root-result G is assigned to c, and c has not yet been observed. Therefore, the system is unable to consider G as an explanation for the observations.

We are working on developing a revised algorithm to address this limitation and consider this a significant area for future work. That said, there are domains where the speed of this algorithm and its ability to allow multiple different choices for plan heads make it worth considering.

8 Conclusions

In this paper, we have defined ELEXIR, a probabilistic plan recognition algorithm using CCGs to encode plans. We have analyzed the complexity of the algorithm, and described its empirical evaluation. We have also shown that CCGs provide a formal way to control the early commitment problem faced by other plan recognition systems.

Acknowledgments

The work described in this paper was conducted within the EU Cognitive Systems project PACO-PLUS (FP6-2004-IST-4-027657) funded by the European Commission.

References

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

[Baldridge, 2002] Jason Baldridge. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh, 2002.

[Bui *et al.*, 2002] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the Abstract Hidden Markov Model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[Carberry, 1990] Sandra Carberry. *Plan Recognition in Natural Language Dialogue*. ACL-MIT Press Series in Natural Language Processing. MIT Press, 1990.

[Clark and Curran, 2004] Stephen Clark and James Curran. Parsing the wsj using ccg and log-linear models. In *ACL '04: Proceedings of the 42th Meeting of the Association for Computational Linguistics*, pages 104–111, 2004.

[Curry, 1977] Haskell Curry. *Foundations of Mathematical Logic*. Dover Publications Inc., 1977.

[Geib, 2004] Christopher Geib. Assessing the complexity of plan recognition. In *Proceedings of AAAI-2004*, pages 507–512, 2004.

[Geib, 2006] Christopher Geib. Plan recognition. In Alexander Kott and William McEneaney, editors, *Adversarial Reasoning*, pages 77–100. Chapman and Hall/CRC, 2006.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Hockenmaier, 2003] Julia Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Catagorial Grammar*. PhD thesis, University of Edinburgh, 2003.

[Kautz, 1991] Henry A. Kautz. A formal theory of plan recognition and its implementation. In James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg, editors, *Reasoning About Plans*, chapter 2. Morgan Kaufmann, 1991.

[Pynadath and Wellman, 2000] David Pynadath and Michael Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of UAI-2000*, pages 507–514, 2000.

[Steedman, 2000] Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

Appendix B

Lexical Ambiguity and its Impact on Plan Recognition.

Christopher W Geib

In submission to ICAPS 2010
Keywords: Plan Recognition, Action Representation

Abstract

Viewing plan recognition (PR) as a parsing problem, this paper distinguishes three sources of ambiguity: *action ambiguity*, *syntactic ambiguity* and *attachment ambiguity*. Previous work PR in has often conflated these different sources of ambiguity. This paper clarifies this distinction and explicitly studies the effect of syntactic ambiguity on the runtime of a particular PR algorithm. It also argues for new method for controlling plan level ambiguity in probabilistic PR based on the idea of plan "heads" using lexicalized grammars.

Introduction

Plan recognition(PR) is the problem of inferring which plans, from a given plan library, an agent is executing based on observations of their actions. PR is a well studied problem in Artificial Intelligence, and has seen a significant increase in interest due to the availability of large quantities of real world sensor data. Following (Carberry 1990) and (Pynadath & Wellman 2000), we are interested in viewing the problem as one of *parsing* a sequence of observations to produce plans.

Starting from real world data and viewing PR as a parsing task, we can see the problem as made up of three major tasks: 1) recognizing *actions*, 2) assigning syntactic *categories* to each of the actions, and 3) combining the actions based on their categories into *plans*. Each of these tasks must address ambiguity that we will refer to as: *action ambiguity*, *syntactic ambiguity*, and *attachment ambiguity* respectively. This paper will focus on syntactic ambiguity, however, to clearly disambiguate this work, we will briefly discuss related research on the other two forms.

Action ambiguity is typically a result of sensor noise. The observation of a single real world action is usually made up of multiple, temporally extended, noisy sensor reports. These reports must be converted into a usable sequence of *observations of actions*. For example, labeling video frames showing an agent reaching for a coffee mug as part of a `grasp-mug` action. This problem is typically called *activity* or *behavior recognition*.

Starting from real world noisy data (video, sonar, passive RF, GPS data and others), successful activity recognition research has used Hidden Markov Models (HMMs) (Bui, Venkatesh, & West 2002), Conditional Random Fields (CRFs) (Liao, Fox, & Kautz 2007; Vail & Veloso 2008), and other forms of Bayesian reasoning (Avrahami-Zilberbrand & Kaminka 2005; Hoogs & Perera 2008; Liao, Fox, & Kautz 2005). In many cases, researchers have shown impressive results with significant variation in the sensor noise. However, even a perfect activity recognizer can not eliminate all ambiguity from the problem.

Consider unambiguously recognizing a `grasp-mug` action. The agent's goal is still unclear. Is the agent going to drink out of the mug? place it on the table? clean it? It is only by considering the larger plans created by sequences of observed actions that we can recognize the goals of the agent.

Previous work in PR has not distinguished between syntactic and attachment ambiguity, however in their work on natural language parsing (Sarkar, Xia, & Joshi 2000)(AXJ) clearly lays out the differences between choosing a syntactic category for a given observation (*syntactic ambiguity*) and finding the correct attachments between the categories (*attachment ambiguity*) that will result in a sentence (in our case a plan).

AXJ show that, in the case of natural language parsing, the computational cost of attachment ambiguity is far less than that of syntactic ambiguity. Their results do not directly transfer to the PR domain due to differences between action grammars and natural language grammars. However, this result suggests exploring the cost of syntactic ambiguity in PR. We leave the study of attachment ambiguity in PR as an area for future work.

We know of no systematic analysis of the effect of varying syntactic ambiguity on PR or specific ways to control it. In this paper, we will first review the PR algorithm (ELEXIR)(Geib 2009) based on parsing plans represented as Combinatory Categorial Grammars (CCGs) (Steedman 2000). We will then discuss how to compute plan level ambiguity. We will then discuss how to use *plan heads* in the CCG representation to control the effects of of plan level ambiguity on ELEXIR's runtime.

ELEXIR Overview

The ELEXIR system(Geib 2009) performs probabilistic PR using a weighted model counting algorithm given a set of *observations* and a *CCG* specification of the plans to be recognized in a *plan lexicon*. To perform plan hypothesis construction, ELEXIR *parses* the observations, based on the plan lexicon, into the complete and covering set of *explanations* each of which contains one or more plan structures. ELEXIR then establishes a probability distribution over the explanations to reason about the most likely goals and plans of the agent. The first step is to encode the plans in CCGs. We refer the interested read to (Geib 2009) for complete details of the formalization and algorithm behind ELEXIR. However, an understanding of how plans are represented in CCGs will be important for our discussion.

Representing Plans in CCG

Consider the simple abstract hierarchical plan drawn as a partially ordered AND-TREE shown in Figure 1. In this

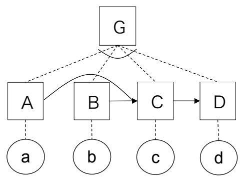

Figure 1: An abstract plan with partial order causal structure

plan, to execute action **G** the agent must perform actions **A**, **B**, **C**, and **D**. **A** and **B** must be executed before **C** but are unordered with respect to each other, and finally **D** must be performed after **C**. To represent this plan in a CCG, each observable action is associated with a set of syntactic *categories*. A set of possible categories, C, is defined recursively by:

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

Complex categories : $\forall Z \in C$, $Z \in C$ 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. Therefore, an action with category $A\backslash\{B\}$ is a function that results in performing action A when an action with category B has already been performed. Likewise, $A/\{B\}$ is a function that results in performing A if an action with category B is executed later.

Definition 1.1 *We define a **plan lexicon** as a tuple* $PL = \langle \Sigma, C, f \rangle$ *where,* Σ *is a finite set of observable action types,* C *is a set of possible CCG categories, and* f *is a function such that* $\forall \sigma \in \Sigma, f(\sigma) \rightarrow C_\sigma \subseteq C$.

where C_σ is the set of categories an observation of type σ can be assigned. We may provide just the function that maps observable action types to categories to define a plan lexicon. For example:

$$a := A, \quad b := B, \quad c := (G/\{D\})\backslash\{A, B\}, \quad d := D. \quad (1)$$

defines one plan lexicon for our example plan.

Definition 1.2 *We define a category* R *as being the **root** or **root-result** of a category* G *if it is the leftmost atomic result category in* G*. For a category* C *we denote this* $root(C)$

Thus, G is the root-result of $(G/\{D\})\backslash\{A, B\}$. Further,

Definition 1.3 *we say that observable action type* a *is a possible **head** of a plan for* C *just in the case that the lexicon assigns to* a *at least one category whose root-result is* C.

In our lexicon c is a possible head for G.

In general, a lexicon will allow multiple categories to be associated with an observed action type. This is the source of syntactic ambiguity s the parser must choose between them.

In CCGs *combinators* (Curry 1977) are used to combine the categories of the individual observations. We will only use three combinators defined on pairs of categories:

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

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 sequence of observations: a, b, c. Notice, all the

$$\begin{array}{ccc} a & b & c \\ \overline{A} & \overline{B} & \overline{(G/\{D\})\backslash\{A, B\}} \\ & \multicolumn{2}{c}{\overline{(G/\{D\})\backslash\{A\}}^{<}} \\ \multicolumn{3}{c}{\overline{G/\{D\}}^{<}} \end{array}$$

Figure 2: Parsing Observations with CCGs

hierarchical structure from the original plan for achieving G is included in c's category. Thus, once c is observed and assigned its category, we can use leftward application twice to combine both the A and B categories with c's initial category to produce $G/\{D\}$.

Empirical Studies of Ambiguity in ELEXIR

Using synthetic grammars and observation streams we can test the impact on ELEXIR's runtime of varying the syntactic ambiguity of the grammar. Constructing plan lexicons and keeping the underlying plan structure fixed while varying the number of observable actions in the grammar provides a simple way to control the number of categories associated with each observable action and the associated syntactic ambiguity.

To see if syntactic ambiguity has a measurable effect even on simple problems, we use totally plans with a tree height of two and a branching factor of five. Thus, each plan has twenty-five steps. For our lexicon we generated sixty-one such unambiguous plans. For each of these plans, the leftmost depth first action of each sub-tree was chosen as the head for the sub-tree. Thus the CCG categories can be thought of as encoded the plan as a series of leftmost depth first sub-trees.

With these CCG categories in hand, we generate multiple lexicons with differing levels of ambiguity by controlling the number of observable actions in the grammar. We measure the ambiguity, A, as a real value between zero and one where the number of observable actions, $|C|$, is given by:

$$|C| = (1 - A) * |l|, \tag{2}$$

where $|l|$ is the number of leaf actions in the plans represented by the lexicon. Given a set of plans encoded in CCGs we can then systematically vary the ambiguity of the resulting lexicon. We, use formula 2 to compute the number of observable actions for the lexicon, given the desired ambiguity, and then randomly assign each category to an observable action while guaranteeing that each observable action gets at least one category. Using this method we generated lexicons for the same set of underlying plans with ambiguities of 0.0, 0.1, 0.2, 0.3, 0.4, and 0.5.

Given these CCG plan lexicons we generated observations to test the system by randomly selecting a root-result categories and producing a plan instance for it based on the plan lexicon. ELEXIR is then timed computing the conditional probability of all the root results found by the algorithm given CCG plan lexicon and the sequence of observations. All of our experiments measuring the runtime for our C++ implementation of ELEXIR were conducted on a MacBook with 4Gb of main memory and 2 2.2-GHz CPUs. Figure 3 shows the average, minimum, and maximum runtimes, testing fifty plan instances each for ambiguity values of 0, 0.1, 0.2, 0.3, and 0.4 with a runtime bound of one minute.

All three statistics show significant increases for even very limited amounts of ambiguity. The reason the maximum and average statistics decrease after 0.2 is that the vast majority of the experiments did not return in under a minute. The number of experiments that did return in under a minute is given along the X-axis in the figure. Once the ambiguity exceeds 0.2 more than half of the test cases jumped from runtimes of under ten seconds to over a minute. As the ambiguity increases the number of successful sub-minute tests drops until none of the tests returned in under a minute when the ambiguity reached 0.5 (and average of two categories per observation).

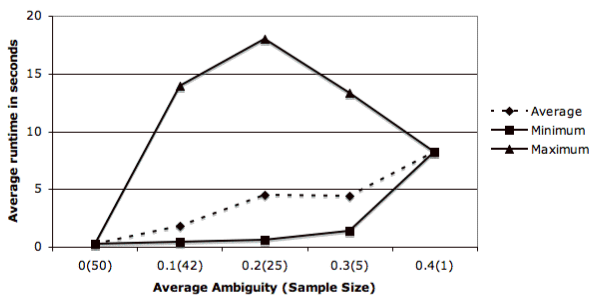

Figure 3: Ambiguity increases min, max and average runtime. Notice the significant ceiling effect above A=0.2

Choosing Heads in Plan Lexicons

The critical choice made by during lexicon construction is which action types will be the plan heads. Different choices for heads result in different lexicons. For example, the following is an alternative lexicon for our G plan.

$$a := A, \quad b := B, \quad c := C, \quad d := (G\backslash\{A, B\})\backslash\{C\}. \tag{3}$$

We can also represent the plan for G with the following lexicon which has two possible categories for action a:

$$\begin{aligned} a := \{ & ((G/\{D\})/\{C\})/\{B\}, \\ & ((G/\{D\})/\{C\})\backslash\{B\} \}, \\ b := B,\ & c := C,\ d := D. \end{aligned} \tag{4}$$

There are also a number of still more complex lexicons where other choices are made for the plans heads. (Geib 2009) has pointed out that correctly choosing plan heads can have significant impact on the runtime of ELEXIR. We hypothesize that correctly choosing plan heads can help in addressing syntactic ambiguity.

It will be helpful to have a value, h, to quantify where the head occurs within a plan. We will establish a canonical order of actions for the plan, that obey the plan's ordering constraints, [1] and define the headedness for a particular plan as the rank of the plan's head action in the ordering divided by the length of the plan. Thus, grammar (3) would have a headedness value of one for the plan for G, grammar (4) would have a headedness value of 0.25 for the plan for G, and our original grammar (1) would have a headedness value of 0.75 for the plan for G.

Reducing Runtimes by Choosing Plan Heads

Our previous experiment held the headedness of plans constant at 0.0. In order to explore the impact that varying headedness might have on controlling ambiguity, we ran experiments systematically varying the headedness of the plans with five values: 0.0 (the same as our previous experiment), 0.25, 0.5, 0.75, and 1.0. Our hypothesis in this experiment is that larger headedness values will delay commitment to high level goals and thereby reduce the runtime of the algorithm.

To create these different lexicons, we used the same set of sixty-one totally ordered plan trees. These plans were then converted to a CCG lexicon by starting at the root of the plan and recursively descending the tree following the actions with the indices given by $\lceil$ (h * *plan-branching-factor*) $\rceil$ collecting siblings that are to the left and the right of the action. When a leaf is reached a CCG category is built maintaining the ordering constraints of the original plan. This process is repeated for all sub plans not covered by the initial category. This results in five grammars where the head of each plan moves from left to right over each of the actions of the plan as the value of h is increased.

Varying both headedness and ambiguity restyled in thirty distinct grammars. For each grammar, we ran fifty tests recognizing a single plan. The minimum runtimes for each of the test conditions is graphed in Figure 4. As we have already seen, placing a one minute bound on the runtime is

[1]For the purposes of our experiments, it will not be significant that actions that are actually unordered with respect to either other can have differing values for headedness. The fact that we can systematically move through the plan's actions is more important.

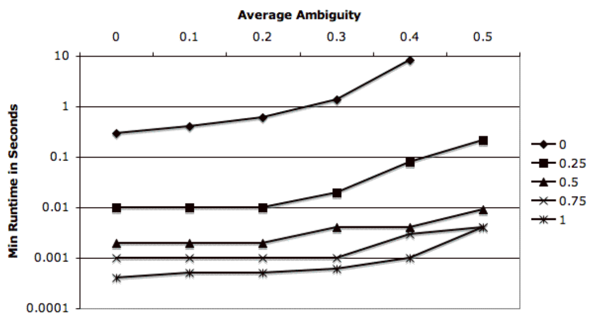

Figure 4: Increasing headedness (moving the head to the right) helps control the cost of ambiguity.

h	A=0.0	A=0.1	A=0.2	A=0.3	A= 0.4	A=0.5
0.01	50	42	25	5	1	0
0.25	50	45	41	39	18	10
0.5	50	45	43	40	36	17
0.75	50	49	37	40	25	25
1.0	50	49	40	39	29	26

Figure 5: Number of test cases with runtimes under one minute.

sufficient to prevent some of the test cases from being completed. Therefore, rather than an average we have graphed the minimum runtimes and remind the reader that these figures represent a lower bound on runtimes for these problems. However keep in mind that in the first experiment that none of the test cases with an ambiguity of 0.5 returned in under a minute.

Figure 4 provides convincing evidence for our hypothesis. We note that each of the lines for the higher headedness values starts with a faster minimum runtime (sometimes two orders of magnitude) and remain below the 0.0 line and even enables many of the test cases for ambiguity 0.5 to return in under one second.

Further evidence of the ability of headedness to aid in controlling ambiguity in plans is seen in Figure 5. This table presents the number of test cases that returned within the one minute bound. It shows that moving the head to the right in a plan increases the number of test cases with a runtime under one minute, relative to plans with the same ambiguity. Thus, even though ambiguity is being increased as we move to the right, increasing headedness in the plans is allowing ELEXIR to run fast enough to return an increasing number of results within the one minute bound.

For example, a headedness value of 0.75 enables half of the tests to return in under a minute where the ambiguity of the plan lexicon had prevented any of the test cases returning when the lexicon had a headedness value of 0.0. Thus we can conclude that not only is the minimum runtime for the algorithm being kept low by moving the head of the plan away from the first actions of the plan, but the number of cases that can be brought within a reasonable runtime is also increasing.

Conclusions

This paper has discussed different sources of ambiguity in the plan recognition. We have provided a systematic study of syntactic ambiguity for a particular PR algorithm. We have shown that even relatively low levels of syntactic ambiguity can be crippling to the runtime of PR algorithms. Finally, we have shown that introducing the idea of heads in plans and moving the heads of plans away from the initial actions of a plan can be a powerful tool to help control the runtime of PR even in the face of significant syntactic ambiguity.

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. Incorporating default inferences into plan recognition. 471–478. MIT Press.

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

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.

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.

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

Liao, L.; Fox, D.; and Kautz, H. 2007. Extracting places and activities from gps traces using hierarchical conditional random fields. In *International Journal of Robotics Research*, volume 26, 119 – 134.

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.

Sarkar, A.; Xia, F.; and Joshi, A. 2000. Some experiments on indicators of parsing complexity for lexicalized grammars. In *Proceeding of the COLING 2000 Workshop on Efficiency in Large-Scale Parsing Systems*.

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 the Conference of the American Association of Artifical Intelligence (2008)*, 1415–1420.