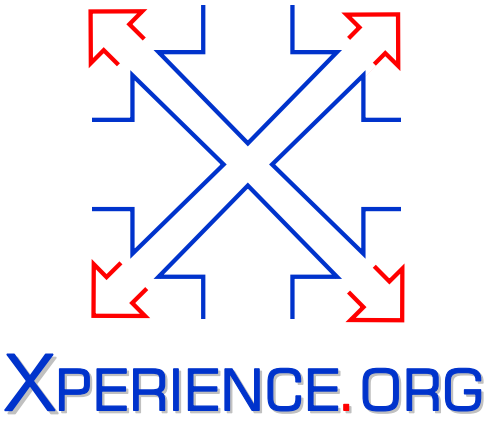
Project Acronym:	Xperience
Project Type:	IP
Project Title:	Robots Bootstrapped through Learning from Experience
Contract Number:	215821
Starting Date:	01-01-2011
Ending Date:	31-12-2015



Deliverable Number:	D4.2.5
Deliverable Title :	Structurally Bootstrapped Plans for Conversational Actions: Report or scientific publication on recognizing conversational plans as critical part of discourse understanding
Type (Internal, Restricted, Public):	PU
Authors:	Christopher Geib, Ron Petrick
Contributing Partners:	UEDIN

Contractual Date of Delivery to the EC:	31-01-2015
Actual Date of Delivery to the EC:	04-02-2015

Contents

2 Introduction and Overview

The link between planning and language has a long tradition, especially with respect to planning for natural language generation and dialogue. Early approaches to generation as planning Perrault and Allen (1980); Appelt (1985); Young and Moore (1994) focused primarily on high-level structures, such as speech acts and discourse relations, but suffered due to the inefficiency of the planners available at the time. As a result, recent mainstream research has tended to segregate task planning from discourse and dialogue planning, capturing the latter with specialised approaches such as finite state machines, information states, speech-act theories, or dialogue games Traum and Allen (1992); Green and Carberry (1994); Matheson et al. (2000); Asher and Lascarides (2003); Maudet (2004).

Recently, there has been renewed interest in applying planning methods to natural language problems such as sentence planning Koller and Stone (2007), instruction giving Koller and Petrick (2011), and accommodation Benotti (2008). The idea of treating interaction management as planning with incomplete information and sensing has also been revisited Stone (2000), a view that is implicit in early BDI-based approaches, e.g., Litman and Allen (1987); Bratman et al. (1988); Cohen and Levesque (1990); Grosz and Sidner (1990). Other work using the knowledge-level PKS planner also explored this connection Steedman and Petrick (2007), but fell short of implementing an efficient tool. A related approach Brenner and Kruijff-Korbayová (2008) manages dialogues by interleaving planning and execution, but fails to solve the problem of deciding when best to commit to plan execution versus plan construction. More recent work has also explored the use of knowledge-level planning for social interaction Petrick and Foster (2013).

In light of this body of work, in the Xperience proposal and in previous deliverables we have argued that Natural Language parsing, and generation, planning, and plan recognition are all processes that should be driven by formal grammars. That is, there is computational leverage to be gained on these problems by viewing them as operations on formal grammars. Further, in earlier deliverables we have demonstrated that language use is best understood as a specialized kind of action, and discourse and dialog planning as special cases of planning for action.

As a result this deliverable will focus on our recent advances in planning and plan recognition based on Combinatorial Categorial Grammars (CCGs) Steedman (2000). In particular it will provide background on CCGs and how they can be used for recognition, then it will cover recient results we have had in extended such grammars to minimize runtimes in domains that have the kinds of properties that language and discourse have. Then it will cover another set of recent results we have had in dealing with selective partial observability of domains. This is again a problem that is particularly common in language domains, where facts and observations are frequently assumed and therefore should be treated as only partially observable. This document will then move on to discuss how the same plan grammars used to recognize plans for physical or communicative actions can be used to build plans for the same plans. It will conclude with a discussion of our ongoing work on bootstrapping of these CCG plan structures taking lessons learned from language learning.

3 Using CCGs for Recognition

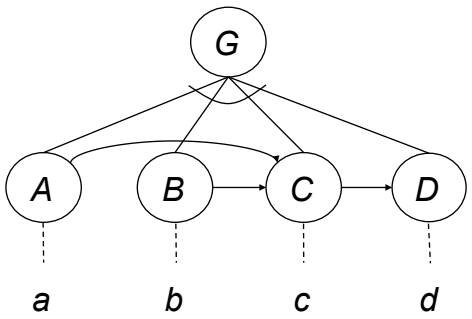


Figure 1: An abstract hierarchical plan with partial order causal structure

We view plan recognition as a kind of parsing. Thus we will assume as given a set of *observations* and

a *CCG* specification of a *plan grammar* or *plan lexicon* defining the plans to be recognized. To perform plan recognition, we will then parse 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 task **G** the agent must perform the sub-tasks **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**. Finally, in order to execute each of the respective sub-tasks the actions a, b, c, d must be performed.

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

Definition 3.1. *We define a set of CCG categories, $\mathcal{C}$, recursively:*

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

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

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 CCG plan lexicon.

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

C_σ is the set of categories an observed action 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,

CCG: 1.

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

defines one plan lexicon for our example plan. For the rest of this paper, we will follow the convention shown here that actions will be written with lowercase script letters (sometimes with subscripts) and basic categories will be in capitals. 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 **anchor** 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 anchor 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 plan recognition 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.

$$\dfrac{\dfrac{\dfrac{\text{a}}{A} \quad \dfrac{\text{b}}{B} \quad \dfrac{\text{c}}{(G/\{D\})\backslash\{A,B\}}}{(G/\{D\})\backslash\{A\}}<}{G/\{D\}}<$$

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 anchors. Different choices for anchors result in different lexicons. For example, the following is an alternative lexicon for G where d is the anchor rather than c.

CCG: 2.

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

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

CCG: 3.

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

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

Modeling issues that are similar to choosing anchors 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 anchors for a CCG this is a design decision for HTNs and can have serious impact on plan recognition and planning algorithms.

Keep in mind, we want to use parsing of CCGs to build explanations for the observed actions to perform plan recognition. However, we don't want to make early commitments to goals. In contrast to traditional HTNs, CCG categories function as a tree and/or sub-tree 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 plan hypothesis we can build and recognize. It enables a least commitment approach, and limits the plan hypothesis to those for which we have observed the anchor of the plan. As we will see next, it also enables a simple algorithm for generating explanations for observations.

Building Explanations

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 plan recognition, 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 plan grammars to only *leftward applicable* categories.

Definition 3.5. *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 3.6. *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.

Computing Probabilities

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 3.7.

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

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 3.8.

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

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 plan recognition. 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 3.8 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, 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 3.8.

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

3.2 Implementation Status

With this discussion of the algorithm and its probability model in hand, we close this section by pointing out that this algorith as discussed has been implemented in C++ in the Engine for LEXicalized Intent Recognition(ELEXIR) plan recognition system, and that an analysis of the complexity of the algorithm can be found in Geib (2009) and further discussion in Geib and Goldman (2011).

4 Anchor Selection

With this model of plan recognition in hand we move on to discussing how to make it more efficient. We note, that as the number of possible plans increases relative to the number of actions, on average each action must play a role in more and more plans. Thus the amount of information provided by each observed action decreases in such domains. Single actions stop being diagnostic of the plan being pursued, and longer and longer sequences of actions are required to recognize the plan being followed.

In such situations, we are doomed to the expensive process of maintaining all of the possible plans that could be going on. This is particularly problematic if it is possible for the agent to be engaged in multiple plans at the same time (a common situation for in language and discourse applications).

However, "reacting appropriately" doesn't always require acting before the plan has completed. Imagine a system designed to recognize three tasks with the following additional constraints. The first plan should be allowed to execute once after which time the system is to change permissions to prevent any further executions of the plan. For the second plan the system only needs to count the number of complete executions. For the third plan it needs to recognize early enough to remind the user about safety critical aspects of its execution. In this case, only one of the plans actually needs to be recognized before it is completed. However, current plan recognition algorithms will attempt to recognize all three of these plans as quickly as possible. Here we advocate a method to reduce the runtime of plan recognition by strategically delaying the recognition of specific plans. In our example, since it is only required that the system be aware of the first two plans once they are completed, two of the three plans could have their recognition delayed in order speed the overall performance of the system.

This is not an argument for delaying the recognition of all plans. In some cases, recognizing the plan too late is as costly as not recognizing it at all. This suggests that reducing runtime for the system as a whole is critical. However, since in the real world not all plans need to be recognized after the smallest number of observed actions, if a speedup can be achieved by delaying the recognition of less critical plans, then we will have improved the system's runtime without any cost to the system's effectiveness.

We can achieve this by differentially encoding critical and non-critical plans in the plan library. The ELEXIR system provides a simple method for controlling when recognition happens for plans in its plan libraries based on changing *anchors* for the plans in the library. By making the anchors for the plan as late as possible in its execution we delay the system's recognition of the plan. By making the anchors as early as possible we force the system to perform early recognition. We will used this method to show that changing the ratio of early recognition plans to late will allow a continuous reduction in the runtime of the algorithm and provide a method for us to control the system's performance.

4.1 Controlling When Parsing Happens

Consider the following example lexicon for four observable actions of getting a cellphone, opening it, dialing a number, and talking. In this lexicon these actions can either be combined to report a fire ($REPORT$) or to talk to a friend ($CHAT$). We will also make use of a small number of other basic categories when needed (D, T, O and G):

CCG: 4.

$$\begin{aligned} dialcell :=& ((REPORT/\{T\})\backslash\{G\})\backslash\{O\} \mid \\ & ((CHAT/\{T\})\backslash\{G\})\backslash\{O\}. \\ talkcell :=& T. \\ getcell :=& G. \\ opencell :=& O. \end{aligned}$$

Requiring the use of minimal lexically justified explanations means that no plan structure can be added that is not introduced by an observed action. This makes the choice of which categories are assigned to which actions a critical design decision. It also means that, since the categories are associated with the observed actions, making different choices about which categories are associated with which actions will have an impact on when different plans are hypothesized. Consider two alternative lexicons for the phone calling example.

CCG: 5.

$$\begin{aligned} getcell :=& ((REPORT/\{T\})/\{D\})/\{O\} \mid \\ & ((CHAT/\{T\})/\{D\})/\{O\}. \\ talkcell :=& T. \\ dialcell :=& D. \\ opencell :=& O. \end{aligned}$$

CCG: 6.

$$\begin{aligned} talkcell :=& ((REPORT\backslash\{G\})\backslash\{O\})\backslash\{D\} \mid \\ & ((CHAT\backslash\{G\})\backslash\{O\})\backslash\{D\}. \\ dialcell :=& D. \\ getcell :=& G. \\ opencell :=& O. \end{aligned}$$

The three lexicons differ in which action is the anchor for the plans for $CHAT$ and $REPORT$. In CCG:4 it is *dialcell* (one of actions in the middle for either plan), in CCG:5 it is *getcell* (the first action of either plan), and in CCG:6 it is *talkcell* (the last action of either plan).

Figures 4, 5, and 6 show us that all three of these lexicons will correctly parse the plans for both chatting with friends and reporting a fire.

$$\dfrac{\dfrac{\dfrac{\overset{getcell}{\overline{G}} \quad \dfrac{\overset{opencell}{\overline{O}} \quad \dfrac{\overset{dialcell}{\overline{((CHAT/\{T\})\backslash\{G\})\backslash\{O\}}} \quad \overset{talkcell}{\overline{T}}}{(CHAT/\{T\})\backslash\{G\}}<}{CHAT/\{T\}}<}{CHAT}>$$

Figure 4: Parsing observations with CCG:4

$$\dfrac{\dfrac{\dfrac{\overset{getcell}{\overline{((CHAT/\{T\})\{D\})/\{O\}}} \quad \dfrac{\overset{opencell}{\overline{O}}}{(CHAT/\{T\})/\{D\}}> \quad \overset{dialcell}{\overline{D}}}{CHAT/\{T\}}> \quad \overset{talkcell}{\overline{T}}}{CHAT}>$$

Figure 5: Parsing observations with CCG:5

$$\dfrac{\dfrac{\overset{getcell}{\overline{G}} \quad \dfrac{\overset{opencell}{\overline{O}} \quad \dfrac{\overset{dialcell}{\overline{D}} \quad \overset{talkcell}{\overline{((CHAT\backslash\{G\})\backslash\{O\})\backslash\{D\}}}}{((CHAT\backslash\{G\})\backslash\{O\}}<}{(CHAT/\{G\}}<}{CHAT}<$$

Figure 6: Parsing observations with CCG:6

However, notice that they produce these results by a different set of combinator applications, and they differ in when the category that contains `CHAT` is introduced to the explanation. They also produce very different results if we use the ELEXIR parsing algorithm to generate the complete set of explanations. Given the same set of observations: ($\{getcell, opencell, dialcell, talkcell\}$) the following set of explanations is produced by each lexicon:

- CCG:4: (4 explanations)

  `[CHAT],[CHAT/{T},T],[REPORT],[REPORT/{T},T]`

- CCG:5: (8 explanations)

```
[CHAT],[ CHAT/{T},T],[(CHAT/{T})/{D},D,T],
[((CHAT/{T})/{D})/{O},O,D,T],[REPORT],
[REPORT/{T},T],[(REPORT/{T})/{D},D,T],
[((REPORT/{T})/{D})/{O},O,D,T]
```

- CCG:6: (2 explanations)

```
[CHAT],[REPORT]
```

Where each bracketed list of categories represents a separate explanation for the observations. Keep in mind that a given explanation can contain multiple plans, therefore each explanation can have multiple categories. For example the second explanation for CCG:4 has two categories, T and $CHAT/\{T\}$. This would be understood as explaining the observed actions with two concurrent plans. One plan to achieve $CHAT$ that is still looking for a T, and one to achieve T alone.

As mentioned in Section 3.1, while explanations like this might seem unusual, they are necessary to guarantee the completeness of the algorithm. Because $CHAT/\{T\}$ is a rightward-looking category it is possible that the next T that is observed is its argument. However, it is also possible that T is the leftward argument for some as yet unseen leftward-looking category, and another T will occur that will act as the argument for the current $CHAT/\{T\}$ category. While it results in a significant increase in the number of explanations, ELEXIR must consider both of these possibilites. Failure to do this would open ELEXIR up to situations where it could find no explanation because over-eager combination had used arguments required for future observations.

This difference in the number of explanations highlights why the choice of anchors in a lexicon has such a significant impact on the runtime of the algorithm. As the number of explanations that must be built and maintained goes up the runtime of the algorithm naturally increases.

As a second example, consider the explanations generated by a partial trace of the plan: $[getcell, opencell, dialcell]$ for each of the lexicons:

- CCG:4: (2 explanations)

```
[CHAT/{T}],[REPORT/{T}]
```

- CCG:5: (5 explanations)

```
[CHAT/{T}],[(CHAT/{T})/{D},D],
[((CHAT/{T})/{D})/{O},O,D],[REPORT/{T}],
[(REPORT/{T})/{D},D],
[((REPORT/{T})/{D})/{O},O,D]
```

- CCG:6: (1 explanation)

```
[G,O,D]
```

Notice that since both of the anchors chosen for CCG:6 are on the last action, there is only a single minimal, lexically justified explanation for the observed actions. Each of the observed actions has been given a basic category and none of the combinators can be used to combine them until the final action of the plan is seen. At this point, the three categories will be used to discharge the arguments for the plans for $CHAT$ and $REPORT$ resulting in the two explanations previously seen.

In contrast, CCG:5 which has anchors on the first action, has five minimal, lexically justified explanations. This results from the introduction of two rightward-looking categories for the first observed action. From then on in the parsing process, anytime a combinator combines this category with a new action's category, ELEXIR has to create another explanation that does not combine the categories.

This might suggest that when building ELEXIR lexicons we should strongly prefer the final action of any plan as its anchor. This minimizes the number of explanations that have to be built and thus reduces the runtime. However, further consideration of the explanations reveals a problem with this approach.

Consider the sole explanation for the CCG:6 case. There is no category in the explanation that has a result of either $CHAT$ or $REPORT$. This means that ELEXIR would incorrectly compute their

conditional probability. Looking back at Definition 3.7, we see that because there are no explanations that have $CHAT$ or $REPORT$ as a result, there would be no explanations in the sum and therefore ELEXIR would compute the conditional probability $CHAT$ and $REPORT$ as zero. ELEXIR must have observed the anchor for a plan in order to be able to correctly compute its conditional probability. This presents a problem for plan recognition and gives us a very good reason to want to place the anchors for the plans as early as possible.

Note that for any complete plan trace, this is not an issue. If the trace is complete, the anchor must be present and therefore the system will correctly infer the conditional probability of the plan. However, as we have already discussed, this is problematic in cases where we are interested in inferring the plans before their final action is observed. To do this with ELEXIR, the anchor for the plan must have been observed. This provides a significant reason to design lexicons with early anchors.

We are left facing a troubling choice, choose early anchors resulting in rightward-looking categories and face possibly unacceptable runtimes, or choose late anchors resulting in leftward-looking categories and possible not be able to recognize the plans early enough to take action.

A Possible Solution and a Hypothesis

However the solution to this problem is in the realization discussed at the beginning of this paper, namely, we don't always care about recognizing ALL of the plans as early as possible. In fact, many plans may only need to be recognized after then have been completely executed. Clearly these plans can be encoded with completely leftward-looking categories. Plans that must be recognized as early as possible must be encoded at rightward-looking categories. However note, these two requirements are not actually in conflict.

Consider the same set of plans. Suppose $REPORT$ is a plan that must be recognized as early as possible (suppose the inferring system wanted to assist by providing extra information like your location to the fire department). In contrast, $CHAT$ is a low importance plans that need be recognized only after it has been completed. It is possible to encode both of these plans while meeting these requirements in ELEXIR even though they are realized by the same sequence of actions. Consider CCG:7:

CCG: 7.

$$\begin{aligned} getcell &:= G \mid ((REPORT/\{T\})/\{D\})/\{O\} \\ opencell &:= O. \\ dialcell &:= D. \\ talkcell &:= T \mid ((CHAT/\{T\})/\{D\})/\{O\}. \end{aligned}$$

In this lexicon the anchors for the two plan have been separated with the anchor for the time critical plan having as an anchor an action that is early in the plan and the less time critical plan having an anchor associated with an action that falls late in the plan. The impact of this is easy to see if we consider the set of explanations that results from observing the first three actions in the plan given CCG:5, CCG:6, and CCG:7.

- CCG:5: (5 explanations)

```
[CHAT/{T}],[(CHAT/{T})/{D},D],
[((CHAT/{T})/{D})/{O},O,D],[REPORT/{T}],
[(REPORT/{T})/{D},D],
[((REPORT/{T})/{D})/{O},O,D]
```

- CCG:6: (1 explanation)

```
[G,O,D]
```

- CCG:7: (4 explanations)

```
[G,O,D], [REPORT/{T}],
[(REPORT/{T})/{D},D],
[((REPORT/{T})/{D})/{O},O,D]
```

In this case, CCG:7 has one less explanation than CCG:5 because the two alternative explanations found in CCG:5 for $CHAT$ have not been constructed. Thus, the lexicon produces more explanations than the all leftward-looking CCG:6 but fewer than the all rightward-looking CCG:5. Exploratory experiments confirm, as we expect from the smaller number of explanations, that CCG:7 has faster runtimes than CCG:5 but slower runtimes than CCG:6.

While this is an encouraging result, the effect might not be true in all lexicons or might not have a significant enough effect to make it worth considering for general use. Disproving these possibilities, lead to the central hypothesis of this recent work, namely that the runtime of plan recognition in ELEXIR can be reduced by careful choice of anchors in the plan lexicon to define a subset of the plans for early recognition and the rest of the plans for late recognition. The next section will discuss how we tested this hypothesis.

4.2 Experiments

In an effort to control for, and avoid possible artifacts relating to other peoples encodings of domains, we chose to built a set of synthetic plans to test the hypothesis. Having ambiguity in the grammar (ie. actions are not uniquely associated with single plans) is critical for modeling real world domains, therefore we chose two experimental factors:

1. the number of actions in the lexicon, and
2. the percentage of leftward-looking plans in the lexicon.

To build lexicons we first created fixed sets of twenty, forty, sixty, eighty, and one hundred distinct actions. For each set of actions we then randomly generated on hundred sequences of actions of the same length. While we ran experiments varying plan length, we will report only on the plans of length six here. These plans were long enough to be of interest but short enough that all one hundred plans in the all rightward-looking lexicons could be run to provide a large number of datapoints. Note that since there were always one hundred plans, as the number of possible actions is decreased the likelihood of an action occurring in more than one plan goes up as does the likelihood of an action occurring more than once in a plans. Thus, the domains with smaller numbers of actions have significant ambiguity within the plans and the lexicon.

Having generated one hundred plans for each of the sets of actions, we then generated eleven lexicons for each such set. For the first lexicon (the 0% case in Figure 7 and Figure 8) all of the plans were encoded as rightward-looking plans. For each subsequent lexicon we increased by ten percent the number of the plans encoded as leftward-looking

Each of the plans in the lexicon was presented as a test input to the C++, multithreaded implementation of ELEXIR (using only a single processing thread) on a Apple MacBook Air running OSX 10.9.1. the results are shown below.

Figure 7 and Figure 8 show the average and worst case runtimes for all one hundred plans in each of the lexicons for the plan length six cases. Note that the y-axes are logarithmic. In all the cases, we can see the average runtime dropping to around .02 seconds. Even the worst case runtimes, while not as smoothly decreasing, show the same downward trend with a significant reduction in runtime with up to twenty percent of the plans being encoded for early recognition.

Notice that even though the selection of plans for leftward-looking encoding was done randomly, the average runtime for the lexicon as a whole drops as the percentage of leftward-looking plans increases. This clearly confirms our hypothesis even for domains with reasonably high ambiguity and provides a strong tool for controlling the runtime of the ELEXIR plan recognition algorithm.

This would suggest, identifying at design time, the smallest possible set of plans that must be recognized as early as possible and coding these plans as rightward-looking and then coding all the other plans within the lexicon as leftward-looking. This should allow the system to recognize critical plans early enough for intervention while keeping the runtime as low as possible.

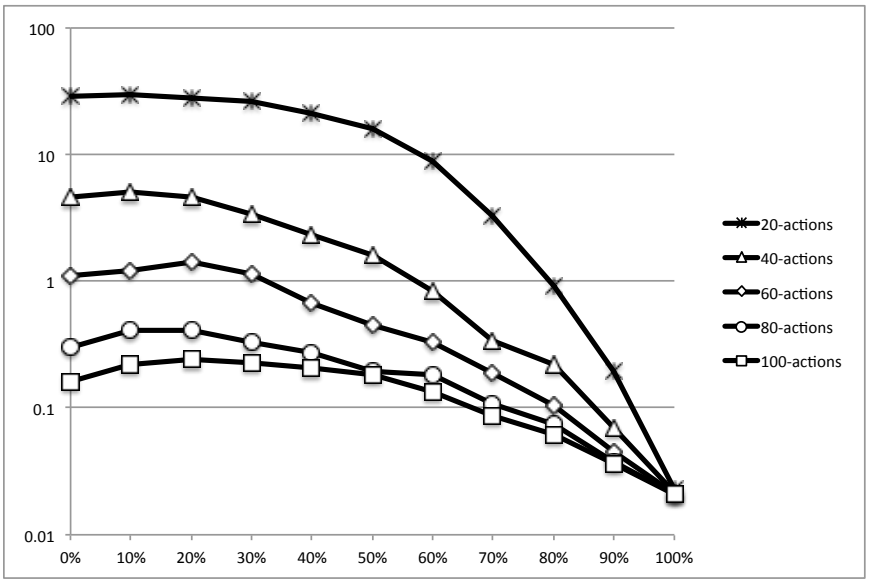


Figure 7: Average runtime in seconds for recognition of 100, 6 step plans vs. Percentage of leftward-looking plans

Impact on Computing Conditional Probability

As we have noted, ELEXIR is a probabilistic system but is unable to compute the conditional probability for any plan for which it has not seen the anchor. This raises the question of the impact of the differential encoding on the computation of probabilities. As we have already said, for complete plans there is no impact. The anchor for the plan will have been observed and thus the conditional probability for it can be computed. It is only in the case of plans that we want to recognize from a partial trace that we have an issue.

Since the critical plans will have been encoded as rightward-looking plans, all of the possible explanations that involve the critical plans will be generated. It is only the explanations for non-critical plans that have not been generated. Thus looking back at Definition 3.7 we can see that while the set of explanations for the critical plans will be present other possible but non-critical explanations may be missing. This has the effect of overstating the probability of the critical-plans. While this is not optimal, for plans that must be recognized as early as possible, an upper bound on the probability is preferable.

4.3 Related Work

Delaying the hypothesis of low importance goals might be thought of as a least commitment approach to plan recognition. Least commitment is a well known way of speeding up computations and has been applied to plan recognition. The work of Avrahami-Zilberbrand and Kaminka (2005) has explicitly attempted to work on least commitment, plan recognition. The early work of Kautz and Allen (1986) can be seen is least commitment. Even Bayesian methods like HMMs (Bui et al. (2002)), CRFs(Liao et al. (2007); Vail and Veloso (2008)), or DBNs(Hoogs and Perera (2008)) are least commitment relative to the information provided.

However, the approach advocated here actually goes beyond least commitment. Note that any minimal, lexically justified explanation is a least commitment explanation for the observations given the lexicon. Only the structures provided by the observations are hypothesized in such explanations. Thus, the ELEXIR parsing algorithm is actually least commitment for computationally expensive rightward-looking grammars.

The approach advocated here actually builds a lexicon that has even fewer explanations than a rightward-looking grammar. Making a different representational choice about the anchors causes the algorithm to have a faster runtime since the search for the plan is limited to ONLY verifying the leftward arguments of the categories. No explanations are generated that will be ruled out by later observations.

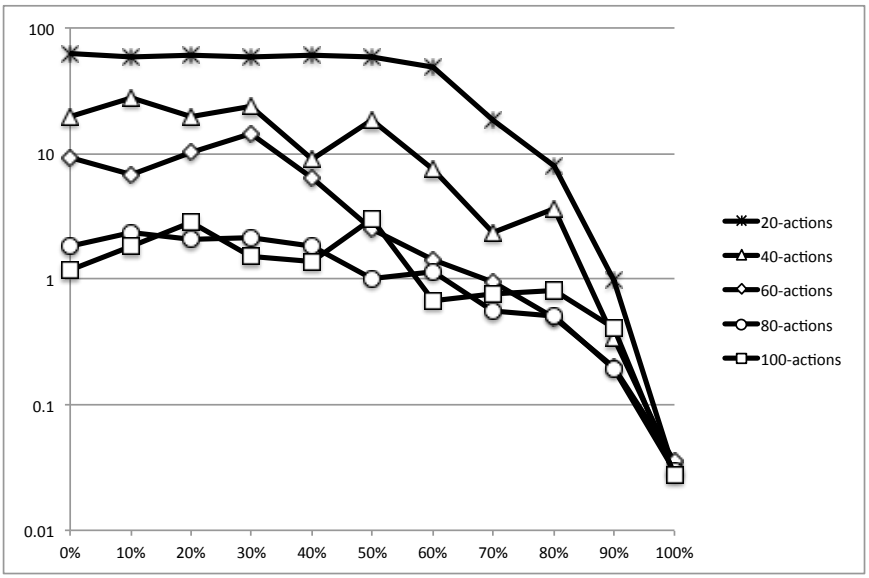


Figure 8: Worst case runtime in seconds for recognition of 100, 6 step plans vs. Percentage of leftward-looking plans

4.4 Implications for Other Methods

This work has shown that differentially encoding plans for early vs late recognition can significantly impact the runtime of plan recognition when viewed as parsing. This gain stems from the incremental nature of the explanation construction process. It is highly likely that other plan recognizers that incrementally develop explanations for observations (based on parsing or other approaches) could benefit from incorporating these idea. For systems that do not take an incremental approach to building explanations for observations the take home message is less clear.

HMMs, CRFs, DBNs, and other traditional Bayesian methods are generally not suited to perform an ordered exploration of the explanation space. These methods consider all of the possible goals in a single model at one time making straightforward application of the ideas presented here difficult. To do so would require considering multiple probabilistic models that explicitly removed the less critical plans from consideration. This would certainly add to the upfront cost of model construction for these methods but might reduce the computation for the individual model evaluation. If this would simplify the models under consideration enough to guarantee a significant reduction in runtime is an open question.

Far more promising for non-parsing based plan recognition, is the prior work on modular Bayes nets of the kind advocated by Pfeffer et al. (1999). Since the model that they advocate is built in an incremental manner, it is possible that ideas advocated here might be incorporated. The open question would again be, if the evaluation of the resulting models was in fact faster than evaluating the full model initially.

4.5 Implementation Status

As this form of controlling of the complexity of plan recognition doesn't require any changes to the baseline ELEXIR algorithm, the work we have described here is a default part of the existing implementation. It is that implementation that was used to generate the results reported above. That said, we are in a process of building software that will help a user make choices to automatically reduce the runtime of an existing ELEXIR lexicon using the ideas reported here. While this kind of modification of the lexicon can be done by hand, automated tools will make it significantly easier and less error prone.

5 Addressing Partial Observability

The ELEXIR system, as discussed so far, has a significant limitation, it assumes that all of the actions in the domain can be observed with certainty. That is, none of the actions can be performed and not

observed or can be incorrectly identified when they have been performed. Thus, ELEXIR requires the sensors used to identify executions of actions are always correct. This is an unrealistic assumption in rototics domains especially when considering discourse.

The fact that ELEXIR is built on a foundation of parsing formal grammars actually provides a very simple method to address this problem. The grammar that captures the plan library can be rewritten to make the observation of the desired actions in the grammar optional, and the probability model of the grammar can be changed to reflect the possibility of the missing observation of the action. This means that no changes are required to the ELEXIR algorithm to deal with partially observable domains. The only change that is required is the rewriting of the plan grammar. As we will see, this does increase the size of the grammar and can therefore have a significant impact on the runtime of the ELEXIR algorithm. The following subsections we will discuss how grammars can be rewritten to account for partially observable actions, and discuss the impact of it on system runtime.

5.1 Partially Observable Actions

To perform plan recognition based on real world data, we need the raw sensor data converted to stream of discrete observed actions. To this end we will assume that one of the many successful *activity recognition systems*(Hoogs and Perera (2008); Liao et al. (2005); Vail and Veloso (2008)) has been used to produce a sequence of such observations. However, activity recognition is fundamentally noisy, error prone, and ambiguous. Thus, there will be times when an action has been executed and the activity recognizer fails to recognize it (quantified by the *false positive rate* for the system), or the activity recognizer believes an action has been performed when in fact it has not (quantified by the *false negative rate* for the system). In fact, this kind of noise is present for all methods of turning a noisy sensor stream into a discrete set of observed actions.

However, since most work on plan recognition assumes a completely observable action stream, the activity streams that are produced by activity recognizers can't be used as input to plan recognition. This work begins to remove the assumption of complete observations for plan recognition. It will do this by allowing individual actions within the domain to be made partially observable. The grammar for the plans that make use of the selected action will be rewritten to make observation of that action optional. The probability model associated with the grammar will also be modified, on the basis of the false negative rate for observing the action, to account for the missing observation.

For this discussion, It will be helpful to be able to write expressions with variables that capture the structure of categories. A vertical bar will be used to describe categories when we don't care about the directionality of the slash operator. Bold, subscripted lowercase "v"s will be used as variables for a single atomic category. So the expression $\mathbf{v_1}|\{B\}$ matches complex categories with any basic category root result that has only a single atomic category B argument on either side.

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

5.2 Making an Action Partially Observable

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

Suppose we wish to make a known action, act_2, partially observable. In effect, we want to create a new grammar whose yields includes all the sequences in our initial grammar, but also includes all of the sequences from the first grammar that contain act_2 with act_2 missing. Thus if $[act_1, act_2, act_3]$ is in the yield of the initial grammar, we need to rewrite the grammar to accept $[act_1, act_3]$ as well. We will do this by adding categories to the lexicon for specific actions.

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

Extending the Grammar's Yield

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

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

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

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

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

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

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

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

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

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

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

As a final note about rule one, this rule is formulated as if each atomic category can occur only a single time in any category definition. However nothing guarantees this and we must address this limitation.

Rule one will in fact have to be invoked within a loop to address the addition of multiple possible c_{new} categories. Again, consider making action act_X partially observable and suppose a category A that is a root result of one of it's categories, c_i^X, occurs n times in a category, c_j^Y, in the definition of action act_Y. For completeness the new grammar must consider all of the possible permutations of act_X being observed and not in any instance of c_j^Y. This will require the creation and adding to act_Y's definition of $2^n - 1$ categories to capture all of the possible non-empty instances of A being accomplished by an execution of act_X and being observed or not. We will return later to discuss the impact of this on the algorithm's runtime.

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

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

Modifying the Grammar's Probabilities

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

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

However, as we have pointed out, c_j^Y could have multiple instances of a category A that could be achieved using unobserved executions of act_X. Therefore we define the probability of a new category more generally as:

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

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

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

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

Remember that, in rule two, the source category's root results is A. In the initial grammar, with some

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

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

Note the two appeals to using uniform distribution in this discussion are an initial approximation. There are a number of more complex possibilites that could be used including conditioning on state or the parse. We also believe the actual distributions should be learnable from real world data, however these more complex models and questions of learnability we leave as an area for future work.

Example

As examples to help in intuitions, consider the small grammar in Lexicon 8 capturing a part of a computer network security domain. This domain has two primary goals: data theft (DT) and denial of service (DOS). In this case, we will show the probability distribution over the categories as a sequence of reals after the category definition of the action.

CCG: 8.

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

Making the action $usr2root$ partially observable with a false negative rate of 0.25 results in the grammar shown in Lexicon 9. Only rule number one is applied in this case because the action in question is not the anchor of top level category. However, this does result in the addition of one category and the changing of the probability distribution for action $remote2loc$.

CCG: 9.

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

If in addition, making the action $synflood$ partially observable with a false negative rate of 0.25, results in Lexicon 10. Rule two is applied here because $synflood$ is the anchor for a top level category DOS. Therefore, we add a new category to action $portscan$ making it an new anchor for $synflood$ when $\$actSix$ is not observed.

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

Figure 9: Runtimes in milliseconds, standard deviation (in parens), and number of explanations (in square brackets)

CCG: 10.

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

In general grammars will be rewritten to handle partially observable actions offline, before plan recognition is performed. Therefore, we will not discuss this algorithm's runtime. However, we do have a complete implementation of the algorithm for rewriting of ELEXIR domains. What is of interest for evaluation is the impact that running the algorithm has on the size of the plan lexicon and the runtime of the ELEXIR system. We discuss this next.

5.3 Experiments

It will be helpful to consider what impact partial observability has on the yield of a grammar in general. Consider a sequence of actions of length m that is in the yield of the grammar and has n instances of an action, act_X, that is to be made partially observable. As we talked about for yield extension rule one, any new grammar that allows act_X to be partially observable, must accept at least $2^n - 1$ new action sequences, one for each possible non-empty subset of the n actions being unobserved. So in order to address partially observable actions the yield of the plan grammar itself must increase substantially.

In the worst case, to achieve this, the number of new categories added to an individual action's definition is $2^n - 1$ where n is the number of instances of a selected argument in the source category. This can represent a significant increase in the size of the grammar, and has the potential to greatly increase the runtime of the system. The $2^n - 1$ increase in the categories can cause a $2^n - 1$ increase in the branching factor for the search for explanations, and the number of possible explanations is the largest determiner of the runtime of the ELEXIR system.

However two factors tend to minimize this impact. First, in practice, n is usually small. In our anecdotal experience values of n that are larger than three are unusual, thus minimizing the size of the increase $2^n - 1$. Second, the increase in the branching factor is limited to those explanations that make use of the actions that have had categories added to them and only those parts of the plan where the action is used. This again has a tendency to minimize the effect of the worst case. As a result, we have found that establishing the efficacy of using this method requires empirical evaluation and can depend on a number of factors.

For example, Figure 5.3 shows average runtimes in milliseconds for ten executions of the same recognition task using both the initial grammar and a grammar modified to deal with a partially observable action for a much larger test domain. The runtimes, standard deviations (in parenthesis), and the number of explanations produced (in square brackets) are shown for three recognition test cases: 1) the observed plan that has no partially observable actions in it (though such actions are present in the domain), 2) a plan that has partially observable actions and their observations are present in the input observations, and 3) a plan that has partially observable actions with missing observations.

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

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

We have already noted that the ELEXIR parsing algorithm minimizes the impact on the runtime in cases where actions effected by the rewriting for partial observability do not play a role in the input sequence. This can be see in the significantly lower runtimes in the first column, the negligible differences in runtime between the two first column cases, and the fact that the same number of explanations are generated in the first column cases.

5.4 Related Work

Kautz' foundational, graph covering based work on plan recognition (Kautz (1991)), in fact did not assume perfect observations, but instead fit the best vertex cover to the plan graph. However, this makes it unable to address a number of issues that ELEXIR does address including multiple interleaved goals. Other early work using logic based reasoning algorithms(Carberry (1990); Litman (1986)), while invaluable for formalizing the kinds of inferences that are necessary for efficient plan recognition. However, being logic based, they were not amenable to extension with probabilities and therefore addressing partially observable domains was not attempted.

As we have already noted, the current very successful work on probabilistic activity recognition using HMMs and CRFs (Hoogs and Perera (2008); Liao et al. (2005); Vail and Veloso (2008)) is probabilistic however it is addressing a different problem than the plan recognition problem. These systems are looking for a single label for a sequence of observations. ELEXIR's objective is to unite a number of such labels into a structured, higher-level plan. Ramirez and Geffner (2011) have proposed using POMDPs that would directly address the partial observability of the world state, however, the task they are solving is different than the task described here. Like the HMMs and CRFs they attempting to infer a single high level goal that is the objective of an agent specified by a POMDP. The work of Bui et al. (2002) presents an interesting alternative to this work in that Hierarchical Hidden Markov Models should be able to address partially observable domains, and capture the hierarchy of plan structure. However, this work is focused on grammar based methods.

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

There has been some prior work on trying to add partial observability to plan recognition based on grammatical methods (Geib and Goldman (2005)), however, there are two significant differences between this work and the previous. First the grammatical formalism that is used in Geib and Goldman (2005) is completely different. That work is based on parsing plan tree grammars and, as such, their parser functions much more like an incremental left to right parser. The ELEXIR method that we have extended here, is able to work incrementally but is not forced to be left to right but instead can build up "islands" of plans and join them together as more evidence is observed making the parser potentially more efficient.

Second and more critically, Geib and Goldman (2005) addresses the problem of partial observability by modifying the parsing algorithm. The parsing algorithm is changed to hypothesize the execution of all actions that could possibly be executed at each time step. This is in addition to maintaining and extending the set of all of the hypothesis that are consistent with the action that is actually observed next in the series of observations. Intuitively, even if almost all of the hypothesized unobserved actions can be ruled out at the first step, the additional computational cost of hypothesizing all of these unobserved actions is significant. Our approach makes no such changes to the parsing algorithm. The process of plan recognition is driven solely by the the observed actions and as such the parsing process remains efficient.

5.5 Implementation Status

As we have pointed out at the beginning of this section, addressing partially observable domains in this manner does not involve modifying the baseline ELEXIR algorithm. Thus, the the work we have described here can be run on the existing ELEXIR implementation. All that is required is the rewriting of the domain files. That said, we have built a piece of software that aids a user in making select actions within a lexicon partially observable. The software reads in a domain, prompts the user for an action to make partially observable and a false negative rate and rewrites the lexicon to take this into account. While this kind of modification of the lexicon can be done by hand, having a tool that performs this rewriting makes the process much less error prone.

6 Planning Using CCGs

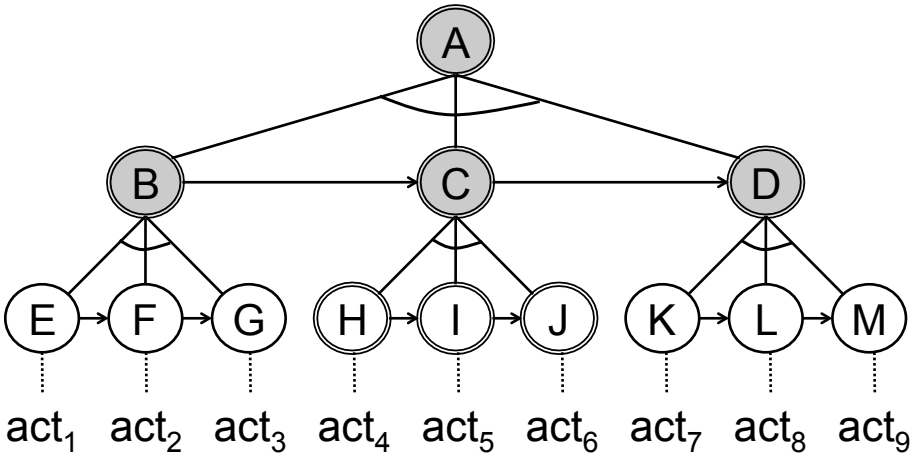


Figure 10: A Hierarchical plan structure with highlighting, Grey nodes are all part of a single HTN method, while nodes with a ringed edge at part of a single CCG category.

Having this understanding of CCGs and how they can be used to recognize plans, we now switch our attention to how we can use the same grammars to direct the building plans to achieve objectives. CCGs can be used to encode hierarchical plan structures like that of Hierarchical Task Networks(HTNs)(Erol et al. (1994); Ghallab et al. (2004). Consider the hierarchical plan structure shown in Figure 10. If we use HTNs to represent such a plan, each decomposition is captured in an *method*. For example a single method would capture the fact that to achieve the highlevel task labeled A the subtasks B, C, and D must be executed in sequence. In an HTN this would be represented with a method that looked something like :

$$A \rightarrow B, C, D$$

and would only convey information about the gray nodes in Figure 10. Likewise there would be a method to define the decomposition of each of B, C, and D. Note that with conventional HTNs none of these methods would make any commitments about how the subtasks themselves should be executed or the order of their further decomposition. It only captures a single level of decomposition from high level task to an ordered sequence of subtasks.

In contrast, as we have already partially described, a single CCG category captures the spine of a tree from the root of the tree (or a subtree) all the way to a basic action at a leaf. For example, if we chose

act_5 as the anchor for a plan to achieve A, then in the lexicon we would expect that it would have a category of the form:

$$act_5 := (((A/\{D\})/\{J\})\backslash\{B\})\backslash\{H\}.$$

Note the portions of the plan tree that are covered by this category are the nodes that have a second ring around them. If we stop thinking of such a category as only the syntax only for recognition of a plan but also think of it as the syntax for the generation of a plan, the category can provide us some very unusual insights to the planning process.

First, note again that lexicalization of the plan grammar places all of the knowledge about how to use basic actions in the categories for the actions themselves. This is why the category for act_5 defines a significantly larger portion of the plan's causal structure than any of the traditional HTN methods. Effectively, knowledge about the plan is being grounded or embodied in the basic actions that are chosen as anchors to achieve high level plans. This is in contrast to HTN planning systems that have to account for the status, organization, and learning of knowledge in the form of methods that is not immediately grounded in the use of the actions themselves.

Second, this category indicates that the basic action act_5 is one of the actions *in the middle* of a plan to achieve A. This is very unusual and as we will see enables a very different kind of method of building a plan.

Third, like the HTN method, it has further information about the subcategories that have the be achieved before and after it. However, we note that some of the subcategories are at what we would consider different levels of granularity in the plan space. For example, B and H in an HTN planner would normally be considered at different stages of the decomposition process. Thus, such a CCG category contains more information about how to achieve A than the HTN method and cross cuts traditional HTN decompositional layers.

To build a planner we will also qualitatively change the way in which we look at the information in the lexicon. While HTNs define a kind of declarative knowledge about the subtasks that are necessary to accomplish a task, this work takes an additional step and interprets the information in the CCGs as a procedural algorithm for building a possible plan to achieve the goal. This is an important distinction. HTNs say nothing about the order in which the subtasks should be further decomposed. Outside heuristics or fixed rules are used to determine the order in which this is done. However past work in HTN planning has shown that such heuristics are brittle, domain specific, and do not scale well.

6.1 The Planning Algorithm

In contrast, our planner uses the directionality and order of the arguments in a CCG category to not only capture ordering relations among the argument subcategories but to also to control when plans to achieve the subcategories are built. Using CCG grammars in this way, suggests that at the same time that we are learning the causal structure of the plan within the grammar, we are also in a position to learn the most effective order to actually expand the argument categories for the plan. This may provide us with further information for how to learn effective CCG categories, specifically how to choose the anchors for a given plan lexicon. That is, information about how to effectively build plans to achieve given categories can help us to choose the anchors for our grammar.

Consider the following simple plan lexicon.

CCG: 11.

$$act_1 := E.$$
$$act_2 := (B/\{G\})\backslash\{E\}.$$
$$act_3 := G.$$
$$act_4 := H.$$
$$act_5 := (((A/\{D\})/\{J\})\backslash\{B\})\backslash\{H\}.$$
$$act_6 := J.$$
$$act_7 := K.$$
$$act_8 := (D/\{M\})\backslash\{K\}.$$
$$act_9 := M.$$

Procedure *BuildPlan*(G) {
  LET $\mathcal{C}_G$ = the set of all $c_i \in \mathcal{C}$ such that $c_i = G|\overrightarrow{\mathbf{v_G}}$;
  FOR $c_i \in \mathcal{C}_G$
    LET a_{c_i} be the action the lexicon assigns c_i to;
    $P = [a_{c_i}]$;
    WHILE $c_i \neq G$
      IF $c_i = \overrightarrow{\mathbf{v_x}} \backslash c_x$
        P = APPEND(BuildPlan(c_x), P);
      END-if
      IF $c_i = \overrightarrow{\mathbf{v_x}} / c_x$
        P = APPEND(P, BuildPlan(c_x));
      END-if
      $c_i = \overrightarrow{\mathbf{v_x}}$;
    END-while
  RETURN P; }

Figure 11: High level recursive algorithm for plan generation.

This is one of the possible lexicons for the plan structure shown in Figure 10 and is consistent with our previous definition of act_5. According to this lexicon the only successful way of building a plan for B that is known by the lexicon is to execute actions act_2. However, this will only be successful if prior to executing act_2 some action, or sequence of actions, that achieves E is executed, and after act_2, an action or sequence of actions must be executed to achieve G. In the case of this lexicon those actions would be act_1 and act_3 respectively. Thus, if we want to use the structure of the category to drive the planning processes, we must first build a plan that commits to performing act_5, then we add to the plan the execution of act_1 that would occurs before act_2. Finally we would add act_3 to the plan after the execution of act_2.

To be more rigorous, consider the pseudocode for building a plan to achieve category C found in Figure 11. This algorithm is really nothing more than a formalization of the ideas sketched above. The order of a complex category's arguments determines the order in which the planning process is performed. Further note that the direction of the slash informs where the subplan is built. Thus this planning process calls for adding actions both to the left and the right of the initial action placed in the plan. In this way, it is similar to causal order planing that was advocated in early planning literature(Penberthy and Weld (1992)) but is not currently in vogue.

Note, that in order to use this algorithm we must be able to select the subset of categories in the lexicon with a particular basic category as its root result, and index back to the basic action that has the complex category in its lexical entry. This sort of index is the only addition to the ELEXIR lexicons that is required to perform this kind of planning. That said it should also be clear that the order in which this list of complex categories is searched will have a profound effect on the speed with which an effective plan can be found, and is a potential target for learning.

6.2 Example

An example may help in clarifying how this algorithm works. Suppose we are interested in generating a plan to achieve category A. Following this algorithm will result in the expected plan made up of all nine actions in order:

$$[\text{act}_1, \text{act}_2, \text{act}_3, \text{act}_4, \text{act}_5, \text{act}_6, \text{act}_7, \text{act}_8, \text{act}_9]$$

this includes building the subplans for B and D. For clarity we have added subscripted brackets to the following list to see the subplan structures within the plan.

$$[[\text{act}_1, \text{act}_2, \text{act}_3]_B, \text{act}_4, \text{act}_5, \text{act}_6, [\text{act}_7, \text{act}_8, \text{act}_9]_D]$$

However, the plan is not built in this order (ie. first to last). Instead it is built in the order prescribed by the grammar. What this means is that actions will be instead into the plan, not from left to right, but instead varying between the left and the right sides of actions that are plan and subplan anchors and have already been added to the plan. To see this, in the following list actions are ordered by their insertion into the plan. We have kept the subplan bracketing for ease of viewing.

$$[\text{act}_5, \text{act}_4, [\text{act}_2, \text{act}_1, \text{act}_3]_B, \text{act}_6, [\text{act}_8, \text{act}_7, \text{act}_9]_D]$$

We note that this truly is a "divide and conquer" style algorithm. Each of the subplans could have been built in any order, but importantly, that order is encoded in the lexicon itself. It is not extra information that must be provided by the algorithm designer, or a heuristic domain specific information. It is information that is specific to the category itself. It is inherent in the lexicon.

6.3 Discussion of Use

After such plans are constructed, the plan must be tested to verify that it achieves the given category. Thus this algorithm is implemented with a backtracking search to allow for the planning process to search for another possible plan if the first returned plan fails to achieve the objective.

It is the intention that the plan lexicon encodes methods for building plans that are *likely* to succeed. They are not guaranteed to succeed but rather each CCG represents a suggestion about how to go about building a plan that might result in achieving the desired goal. This is in contrast to most HTN planners that assume the specified set of plans are complete. Thus in this style of planning, the search for a plan to achieve a particular goal category is informed by the list of categories that have that basic category as their root result. This set of categories is not guaranteed to always contain a successful plan. This is the reason for the simulation of the completed plan, namely to verify its correctness.

Thus, like Fast Forward (FF) Hoffmann and Nebel (2001) and other state of the art planners, this planning method is incomplete. We do not claim that the lexicon is the definitive collection of all of the possible methods of achieving some category. So like FF and related planners, if completeness is required for the domain, and the lexicon does not define a plan, then we must fall back on forward search through the space of basic actions, or a similar method.

It could be suggested that a more traditional HTN could be augmented with the same kind of interleaved expansion of the subplans by simply providing subgoal planning guidance. However crucially, it should be clear that this approach is superior since the lexicon itself defines this ordering. No fixed algorithm for this could be as flexible. The anchors chosen for the plans determining the order in which actions are planned. So we can have much more complex plan building orders by careful choice of anchors. Even if the lexicon obeys the requirement for only using leftward applicable grammars, the choosing of different different subgoals and different anchors for those subgoals at the time of lexical acquisition can have a significant impact on the ordering of the building of plans.

6.4 Implementation Status

A full, first-order logical planner that implements these ideas and uses ELEXIR lexicons has been built in C++. It was built using the same core data-structures and representations as the ELEXIR recognizer. We are in the process of performing benchmark evaluations of this planner against existing publicly available implementations of HTN planning in the form of the SHOP2 planner(Nau et al. (2003)) using domains taken from the International Planning Competition(http://ipc.icaps-conference.org). We anticipate similar performance levels. However, we believe that the close relation between this planner and the ELEXIR system, namely their use of common underlying data structure and a lexicon formalism taken from computational linguistics, is an intellectual contribution the would ameliorate even moderate decreases in performance relative to other state of the art planning systems.

7 Bootstrapping CCG Plan Grammars

So far we have discussed the ELEXIR plan recognition system and a planning system that is directed by the same CCGs. In this penultimate section we will discuss our ongoing work on learning plan CCGs. For this discussion, it is worth noting that we distinguish the learning of plan grammars from learning of actions. There is a significant body of literature on learning actions, however this work will not address this issue. Instead, this work assumes a set of basic actions as given and we are proposing to learn the grammar, or rules, that govern the acceptable sequences of actions that will result in the achievement of a particular goal on the basis of those actions. Before discussing our proposed method for learning CCG categories it will be helpful to discuss some of the assumptions of previous research in learning HTNs and natural language CCGs, and why this prior work is not directly applicable here.

7.1 Background on Learning Hierarchical Transitions Networks

At this point it should be clear that CCGs are certainly able to express the kinds of plan decomposition that are normally captured in HTNs. As such, a natural question to ask is can we use existing research on learning HTNs for methods to learn our CCGs? Two of the best known pieces of prior work on this topic are that on HTN-MAKER(Chad Hogg (2008)) or WIT(Fusun Yaman (2009)).

Unfortunately both of these methods fall sort of solving the problem we are interested in solving. While HTN *methods* that describe decomposition do bear a striking resemblance to context-free grammar(CFG) rules, they are usually presented as having preconditions and post conditions (two things that normal CFG rules and CCG categories do not have). In fact, much of the effort in both HTN-MAKER and WIT are devoted to recognizing these invariant conditions. This is not part of learning CCGs and so seems less directly relevant to our objectives. Further, while HTN-MAKER is in theory able to produce left branching and right branching grammars, there are still open questions about how and when the grammar should branch in which direction, and the implementation assumes that only a single direction will ever be learned. As we will see this is a significant issue for learning CCGs.

In contrast to HTN-MAKER, WIT's method for identifying hierarchy within the plan focuses on branching and merging points within the causal structure of the plan. While this is one form of abstraction that is possible, it is not the only one. As a result the expressiveness of the grammars that WIT is able to learn is significantly less than that expressible in CCGs. In fact WIT is limited to class of regular grammars while CCGs are slightly stronger than CFGs(Schabes (1990)).

Finally its worth noting that, to the best of our knowledge, HTNs have never been directly applied to parsing natural languages. In this work we are attempting to use a formalism and learning methods taken directly from NLP to learn plan grammars in order to demonstrate a unified understanding the fours problems. If we modify the representations or use method that have not been demonstrated in both domains we weaken the strength of the intellectual contribution of the research. This said, HTN-MAKER, WIT, and other work on learning HTNs are very interesting pieces of work, very much in the spirit of our research program on learning CCGs.

7.2 Background on Learning NLP Grammars

There is also a large body of work on learning NLP grammars. Having settled on CCGs for our gramatical formulation, we will not review the large body of literature on learning other gramatical forms. We do this mostly because the majority of this work has not focused on lexicalized representations and we believe that grounding plans with this formalism provides a significant source of leverage in the learning problem.[1] Instead we will focus on prior successful methods for learning CCGs.

In almost all prior work on learning CCGs(Kwiatkowski et al. (2012); Hockenmaier (2003); Clark and Curran (2004)), has worked in roughly the same manner. The space of possible categories is enumerated and a probability distribution is learned over the set of possible categories for each word that is observed. This method is based on the assumption that the space of possibly categories can be enumerated beforehand.

[1] All the categories that are to be learned must be associated with an individual action and so every use of the action to achieve the plan in question should provide some evidence for or against any category hypothesis that we have.

This is not an unreasonable assumption to make in NLP domains. We have good reasons to believe (from linguistics and psychological studies) that people actually make use of gramatical classes like: nouns, verbs, adjectives, determiners, etc.. Thus, it is not unreasonable to think that we could enumerate a covering set of such types, decide on the CCG category for each according to the regularities of the language and then our task is only to learn the distribution over this set. However, this assumption is much more troublesome in the space of plans.

It is certainly not obvious what the covering set of possible plan types would be. In the space of planning and acting what would correspond to nouns, verbs and the like? Even if we imagine some kind of Shank and Abelson (1977) style prototypes that might try to enumerate the basic abstract interactions going on in plans (moving some thing, acquiring some thing, giving some thing to someone, and the like), we would never want to suggest that such a set was closed and finite. Thus we could not easily learn the distribution over such a set.

To answer this one could imagine learning language using Chinese restaurant, or Indian buffet processes(Pitman (2006)). Such a methods allows for building a statistical models where in there is always a possibility of a new category being added to the distribution. Thus no closed fixed set of plan types would have to be enumerated before hand. This is a possibility and we leave its exploration as an alternative research direction if our current work with chart based methods do not prove fruitful.

As we will see, the work that we will be basing our approach on (Thomforde and Steedman (2011)), doesn't assume a known fixed set of base types. Instead this approach builds a new category from the existing ones. This raises a question about where the initial categories come from but does not require that there be only a small fixed number of them.

7.3 Learning plan CCGs by chart parsing

We now move on to discuss how we propose to learn CCG action categories. In language learning for previously unseen words, Thomforde(Thomforde and Steedman (2011); Thomforde (2013)) has shown that traditional chart parsing, of the kind done by the well known CKY parsing algorithm(Younger (1967); Sipser (1997)), enables the production of CCG categories. That is, we can build categories for portions of the parse of a natural language sentence that do not contain a new word. On the basis of this parse of the known words in a sentence, a category for a previously unseen word can be pieced together. This is the inspiration for our approach. Rather than giving a detailed example of CKY and Thomforde's work we will provide the following illustrative example, and then discuss some limitations and open research questions we are in the process of exploring.

Consider the following example CCG lexicon:

CCG: 12.

$$act_1 := A, \qquad act_2 := B, \qquad act_3 := (((G/\{E\})/\{D\})\backslash\{A\})\backslash\{B\}, \qquad act_4 := D, \qquad act_5 := E$$

Suppose we are presented with a set of five observed actions $[act_1, act_2, act_3, act_4, act_5]$. The lexicon admits the parse that these are all a single instance of achieving the goal G (shown in Figure 12) with all of the structure for the plan being provided by act_3's category.

Now suppose that instead we observe a new action act_6 used in the following sequence: $[act_1, act_2, act_6, act_4, act_5]$ and further, either through observation of a state change or by being informed by an external source, we know this sequence also achieves G. Intuitively, the simplest algorithm we can think of to infer a category for act_6 would be to assign each known observation its most likely category and then walk the resulting parse and build up a new category for the new observation that simply takes each of the known observations categories as an argument (either to the left or the right).

This results in:

$$act_6 := (((G/\{E\})/\{D\})\backslash\{A\})\backslash\{B\}$$

This is very similar to Thomforde's approach and is also reminiscent of explanation based learningSamuelsson (1994).

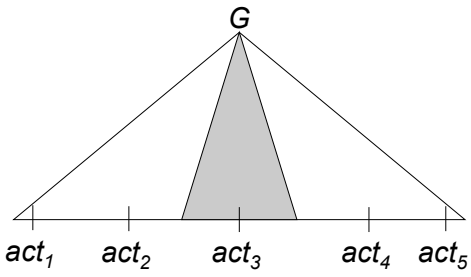


Figure 12: An abstract recognized plan for the goal G

In fact, this process yields the exact same category as act_3 which corresponds nicely with our intuitions. The observed sequences are otherwise the same and achieve the same goal. It seems obvious to add act_6 to the lexicon with the same category as act_3.

We have implemented exactly this simple algorithm within the ELEXIR system for very simple cases, and have shown that such categories can then be used for recognition. However, this line of reasoning hides a number of assumptions and our intuitions seem buttressed by the same assumptions. Removing these assumptions is the next step of our research program. We briefly discuss a few of these next in rough order of their difficulty.

Multiple categories for the surrounding observations: We have strong intuitions about this example because each of the observations only has a single possible category. However, suppose that each of them had two possible categories. How should we choose between them when building the category for act_6? In the example (following Thomforde) we suggested using the most likely one, but is this actually the correct course? While this worked reasonably well for learning low frequency words in English, perhaps in plan grammars choosing a category with a less likely prior would eventually result in a more accurate grammar for other plans.

Not the anchor: The algorithm sketched above assumed the new action would be the anchor for the plan. Is this a warranted assumption? Especially in domains with significant prior knowledge, this may lead to very inefficient grammars where in each observation has a large number of very specialized categories. Instead it may be far more effective to have the new observation take on a known category for a subgoal from another observation. How would we find this existing category or choose between multiple if there exists more than one viable parse for the whole plan?

Previously seen observations: One of the reasons our intuitions are strong in this example is that act_6 has never been seen before. Suppose that act_6 were well known and already had it had a large set of possible categories for it. Perhaps, we might prefer not adding a category for act_6, but instead adding a category to another observation, say act_4 to extend the lexicon to cover this plan for G.

Multiple new observations: Suppose instead of a single new observation we see a sequence of them. How should we assign categories to them? Everything that we have said about the single observation case applies to this case as well. However, since there are multiple observations the categories that should be assigned to each of the individual observations is less clear. There may be a number of ways in which the structure of the subsequence can be produced, and identifying the most effective such is a difficult problem in its own right.

New root goals Suppose we are still faced with a single previously unobserved action, but we are also told that the sequence of actions achieves a previously unknown goal, say G'. While the lexicon may contain significant knowledge about each of the known observations, we are much less certain about which of these categories should be be assigned to each of the observations and how they should be combined to produce the new goal.

We believe that some of these questions can be addressed by making reasonably aggressive additions to the lexicon to allow for immediate recognition of future instance of the plan and then allowing updates

of the lexicon's probability model over time to remove (or at least make arbitrarily less likely) those categories that don't capture the regularities we want in our plan grammars. That said these questions must be addressed if we are to effectively learn CCG plan grammars and addressing them is the core of our ongoing work on Xperience.

7.4 Implementation Status

As we have already alluded, the simple chart based category inference has been implemented in the ELEXIR system. Further, we have demonstrated the ability to learn a single category and then use it to recognize later instances of the same plan. We are in the process of building larger scale test suites to help us to explore how and when this approach will be sufficient for our needs in Xperience. We are also continuing to experiment with this approach to learning and are looking at issues of scaling and addressing the research questions outlined above.

8 Conclusion

As we have argued, the link between planning and language has a long tradition, especially with respect to planning for natural language generation and dialogue. One corner stone of the Xperience project's motivation has been to draw together these two research areas by the use of common representations and algorithms. Further, we have strongly endorsed the research program that language use, dialog and conversational actions are first and formost *actions* of the same basic kind as physical actions. As such, we have engaged in an ongoing program to integrate planning and plan recognition with natural language formalisms.

As a part of this, this deliverable has outlined our existing research prototypes in planning and plan recognition. It has further discussed our recent research results in extending these shared, underlying grammatical formalisms and algorithms to handle both partially observable domains and controlling the runtime of the recognition algorithms. Both of these results are necessary if the plan recognizer and planner are to be used for discourse and conversational actions. This deliverable has gone on to outline the current state of our research on learning new plan grammar categories by bootstrapping from known plan grammars in the form of chart parsing. Finally, it has further outlined some of the open research questions that we are pursuing that are necessary for this work to fully encompass the vision of a single shared framework that is able to recognize and plan for both physical and conversational actions.

References

Appelt, D. (1985). *Planning English Sentences*. Cambridge University Press.

Asher, N. and Lascarides, A. (2003). *Logics of Conversation*. Cambridge University Press.

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

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Benotti, L. (2008). Accommodation through tacit sensing. In *Proceedings of SemDial-2008 (LONDIAL)*, pages 75–82.

Bratman, M., Israel, D., and Pollack, M. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.

Brenner, M. and Kruijff-Korbayová, I. (2008). A continual multiagent planning approach to situated dialogue. In *Proceedings of SemDial-2008 (LONDIAL)*, pages 67–74.

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.

Chad Hogg, Hector Munoz-Avila, U. K. (2008). Htn-maker: Learning htns with minimal additional knowledge engineering required. In *Proceedings of AAAI 2008*, pages 950–956.

Clark, S. and Curran, J. (2004). Parsing the wsj using ccg and log-linear models. In *ACL '04: Proceedings of the 42th Annual Meeting of the Association for Computational Linguistics*, pages 104–111.

Cohen, P. and Levesque, H. (1990). Rational interaction as the basis for communication. In *Intentions in Communication*, pages 221–255. MIT Press.

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

Erol, K., Hendler, J. A., and Nau, D. S. (1994). Htn planning: Complexity and expressivity. In *Proceedings of AAAI-1994*, pages 1123–1128.

Fusun Yaman, Time Oates, M. B. (2009). A context driven approach for workflow mining. In *Proceedings of IJCAI 2009*, pages 1798–1803.

Geib, C. and Goldman, R. (2011). Recognizing plans with loops represented in a lexicalized grammar. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*, pages 958–963.

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)*, pages 1702–1707.

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

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

Green, N. and Carberry, S. (1994). A hybrid reasoning model for indirect answers. In *Proceedings of ACL-1994*, pages 58–65.

Grosz, B. and Sidner, C. (1990). Plans for discourse. In *Intentions in Communication*, pages 417–444. MIT Press.

Hockenmaier, J. (2003). *Data and Models for Statistical Parsing with Combinatory Catagorial Grammar*. PhD thesis, University of Edinburgh.

Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:2001.

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)*, pages 1551–1554.

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

Kautz, H. and Allen, J. F. (1986). Generalized plan recognition. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI-86)*, pages 32–38.

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., editors, *Reasoning About Plans*, chapter 2. Morgan Kaufmann.

Koller, A. and Petrick, R. P. A. (2011). Experiences with planning for natural language generation. *Computational Intelligence*, 27(1):23–40.

Koller, A. and Stone, M. (2007). Sentence generation as planning. In *Proceedings of ACL-2007*, pages 336–343.

Kwiatkowski, T., Goldwater, S., Zettlemoyer, L. S., and Steedman, M. (2012). A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. In Daelemans, W., Lapata, M., and Màrquez, L., editors, *EACL*, pages 234–244. The Association for Computer Linguistics.

Liao, L., Fox, D., and Kautz, H. (2007). Extracting places and activities from gps traces using hierarchical conditional random fields. *International Journal of Robotics Research*, 26(1):119 – 134.

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

Litman, D. (1986). Understanding plan ellipsis. In *Proceedings of AAAI-86*, pages 619–626.

Litman, D. and Allen, J. (1987). A plan recognition model for subdialogues in conversation. *Cognitive Science*, 11:163–200.

Matheson, C., Poesio, M., and Traum, D. (2000). Modeling grounding and discourse obligations using update rules. In *Proceedings of NAACL-2000.*

Maudet, N. (2004). Negotiating language games. *Autonomous Agents and Multi-Agent Systems*, 7:229–233.

Nau, D., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404.

Penberthy, S. J. and Weld, D. S. (1992). Ucpop: A sound, complete, partial order planner for ADL. In Nebel, B., Rich, C., and Swartout, W., editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. Morgan Kaufmann, San Mateo, California.

Perrault, C. R. and Allen, J. F. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, 6(3–4):167–182.

Petrick, R. P. A. and Foster, M. E. (2013). Planning for social interaction in a robot bartender domain. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2013), Special Track on Novel Applications*, pages 389–397, Rome, Italy.

Pfeffer, A., Koller, D., Milch, B., and Takusagawa, K. (1999). Spook: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the Fifteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 541–550, San Francisco, CA. Morgan Kaufmann.

Pitman, J. (2006). *Combinatorial Stochastic Processes.* Springer-Verlag, Berlin.

Pynadath, D. and Wellman, M. (2000). Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 2000 Conference on Uncertainty in Artificial Intelligence*, pages 507–514.

Ramirez, M. and Geffner, H. (2011). Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 3, pages 2009–2014. AAAI Press.

Samuelsson, C. (1994). *Fast Natural-Language Parsing Using Explanation-Based Learning*. PhD thesis, Swedish Royal Institute of Technology (KTH).

Schabes, Y. (1990). *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, University of Pennsylvania.

Shank, R. and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Associates, Hillsdale, N.J.

Sipser, M. (1997). *Introduction to the theory of computation*. PWS Publishing Company.

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

Steedman, M. and Petrick, R. P. A. (2007). Planning dialog actions. In *Proceedings of SIGdial-2007*, pages 265–272.

Stone, M. (2000). Towards a computational account of knowledge, action and inference in instructions. *Journal of Language and Computation*, 1:231–246.

Thomforde, E. (2013). *Semi-supervised Lexical Acqusition for Wide-coverage Parsing*. PhD thesis, University of Edinburgh.

Thomforde, E. and Steedman, M. (2011). Semi-supervised ccg lexicon extension. In *EMNLP*, pages 1246–1256. ACL.

Traum, D. and Allen, J. (1992). A speech acts approach to grounding in conversation. In *Proceedings of ICSLP-1992*, pages 137–140.

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)*, pages 1415–1420.

Vilain, M. (1990). Getting serious about parsing plans. In *Proceedings of the Conference of the American Association of Artifical Intelligence (1991)*, pages 190–197.

Young, R. M. and Moore, J. D. (1994). DPOCL: a principled approach to discourse planning. In *Proceedings of the International Workshop on Natural Language Generation*, pages 13–20.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 2(10):189–208.