Lexicalized Reasoning

Christopher Geib CGEIB@DREXEL.EDU
College of Computing and Informatics, Department of Computer Science, Drexel University, 3141 Chestnut Street Philadelphia, PA 19104 USA

Abstract

The paper argues for the use of lexicalized grammars, specifically Combinatory Categorial Grammars(CCGs), to direct both recognition and generation problems in planning domains. It reviews previous work on using CCGs for plan recognition and then outlines a new algorithm for planning that makes use of the same grammars used for recognition. It does this as part of a larger program to establish and leverage linkages between reasoning about action and language understanding.

2. Introduction and Overview

Research in artificial intelligence in the last fifty years has made significant strides by examining subproblems rather than attempting to build integrated systems. However, it has often been noted that there are close relationships between the identified subproblems. For example, the relationship between natural language parsing and plan recognition (recognizing the plans of others based on observations of their actions) is well known (Vilain, 1990; Pynadath & Wellman, 2000), as is the relationship between natural language sentence generation and planning(Carberry, 1990; Petrick & Foster, 2013). However, while we often pay lip-service to these relationships, we rarely inform our research programs based on the results and methods of other programs. This paper presents a research program that begins to take these interrelations seriously.

Many "AI Hard" problem areas seem to naturally contain a *recognition problem* and a *generation problem*. For example, parsing natural language sentences, and generating novel natural language sentences are clearly related. Likewise the problem of plan recognition and building our own plans seem intimately intertwined. While the same grammars have been used for parsing and generating sentences of natural language, we know of no work that used the same knowledge structures to both build plans and recognize the plans of others. Note, we are not suggesting that the same approaches haven't been used in past work. Multiple other representations including Hidden Markov Models(Baum et al., 1970) and Hierarchical Task Networks(Erol, Hendler, & Nau, 1994) have been proposed as representations for both planning and plan recognition. However, we know of no work that has made the stronger commitment to using the exact same HMM or HTN for both tasks. If our eventual objective is to build systems that integrate multiple subsystems, we believe, there is much to be

learned by requiring the sharing of structures between the tasks. In this paper, we will present just such a pair of systems. Further, in an effort to bridge the gap between the domains of reasoning about language and reasoning about action, the planning and plan recognition systems shown here will be based on a formal grammar model taken from current research on natural language processing. Specifically, this work will build on prior work(Geib & Goldman, 2011) using Steedman's Combinatory Categorial Grammars (CCG)(Steedman, 2000) for plan recognition, that has shown success producing state of the art runtimes while addressing a number of open issues from earlier work.

CCG's are part of a recent trend in natural language processing of *lexicalization* of grammars(Schabes, 1990; Joshi & Schabes, 1997). In lexicalized grammars, all domain specific information is moved out of grammar rules and into the lexicon. Thus, domain knowledge is coded into rich functional categories that are associated with the individual terminals of the grammar by a grammar lexicon. In the case of natural language the grammar terminals would be individual words, while for plan grammars, the terminals would be basic, executable actions. These categories are then combined by domain independent rules either to parse sequences of terminals for recognition or to generate new sequences of terminals.

As we will see, lexicalization, has the benefits of isolating all domain dependent information needed for the reasoning task in the lexicon, allowing for a domain independent implementation of both recognition and generation processes, and precompiling search that would otherwise have to be duplicated. We will use the term *lexicalized reasoning* to capture the kinds of reasoning processes that can be directed by lexicalized grammars. Since lexicalization has already proved successful in natural language parsing and generation they are the two problems first solved by lexicalized reasoning systems.

This paper will focus on two more problems solvable by lexicalized reasoning. Namely, how CCGs can be used to reason about actions. It will outline how CCGs have been used for plan recognition and then discuss new work capturing how the same CCGs can be used to direct plan generation. This will present the first work we know of to use not just a similar, but an identical, discrete action representation for both recognition of plans and their generation based on a representation taken from NLP.

Thus while the central contribution of this paper will be the discussion of a novel planning algorithm, the central contribution for cognitive systems should be seen in the much larger canvas of the lexicalized reasoning research program. This programs central object is to produce two fixed algorithms, one for generation and one for recognition, that operate on a common grammar framework, such that only by varying the input grammar will allow the solving of four different "AI hard" problems, namely language parsing and generation and plan recognition and generation. As such, lexicalized reasoning will successfully address four well known problems by using a common representational framework. Further this kind of relationship is exactly what we would imagine if language use and reasoning about physical action leverage the same cognitive infrastructure, as has long been suggested(Lashley, 1951; Miller, Galanter, & Pribram, 1960).

The rest of this paper will be organized in the following manner. First, it will review how CCGs can be used for plan recognition, then it will discuss new work that details how CCGs can be used to direct plan generation and the requirements for such a grammar and system. It will then close with a discussion of the implications of this approach and outline directions for future work.

3. Using CCGs for Recognition

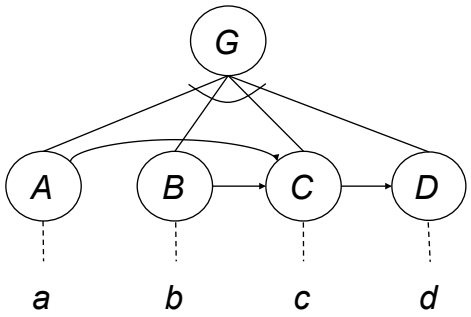

Figure 1. An abstract hierarchical plan with partial order causal structure

We can view plan recognition as a kind of parsing. 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. At a high level, to perform plan recognition, we will 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 will 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, **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 $Z/\{W, X, ...\} \in \mathcal{C}$.*

To formalize actions as functions, we will think about atomic categories are zero arity functions that result in their atomic category. Complex categories are functions that take a set of *arguments* ($\{W, X, ...\}$) and produce a *result* (Z). The direction of the slash indicates where the function looks for its arguments. We require the argument(s) to a complex category be observed after the category for forward slash, or before it for backslash.

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.

To ground CCGs in possible states of the world, we will associate with each atomic category a possibly disjunctive set of *satisfaction conditions* that define those states of the world in which the category is said to be satisfied. Thus, if the satisfaction conditions for

category A are $P \vee (Q \wedge \neg R)$ (where P, Q and R represent propositions that can be true or false in the world) and the lexicon specifies $a := A$, and $c := A \backslash \{B\}$, then we can interpret action a as a function that unconditionally results in states where $P \vee (Q \wedge \neg R)$ is true. Likewise, c is a function that results in the same kind of states, but only if immediately before its execution, states that satisfy B's satisfaction conditions are met. While satisfaction conditions ground our CCGs in possible world states, their importance will be much greater when we consider planning. Thus we will leave further discussion of them to Section 4.

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:

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

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. As each

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

Figure 2. Parsing Observations with CCGs

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 this discussion, we have avoided some of the representational questions in designing a plan lexicon. The critical choice made during lexicon construction is which actions will be 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.

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

Or we could represent the plan for G with a lexicon where a is the anchor and has two possible anchor categories for G that differ in B's position relative to it:

CCG: 3.

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

Modeling issues that are similar to choosing anchors for CCGs occur in traditional hierarchical task network (HTN) representations (Ghallab, Nau, & Traverso, 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 subtree spine crossing multiple levels of plan decomposition. We can use the "vertical slicing" of plans by categories to define the scope of our commitments in building goal and plan hypotheses. We state the following principle:

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

This principle clearly defines when, how much, and what kind of plan structures and 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.

3.2 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 review the algorithm for parsing action categories into explanations given in (Geib, 2009).

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 observed action instances $\sigma_1 ... \sigma_n$ 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. 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 first check in the algorithm.

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.

3.3 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 & 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.

We close this section by pointing out that this algorithm has been implemented in the Engine for LEXicalized Intent Recognition(ELEXIR) plan recognition system, and that more complete discussion of the algorithm, probability model, and an analysis of the complexity of the algorithm can be found in (Geib, 2009) and (Geib & Goldman, 2011).

4. Planning Using CCGs

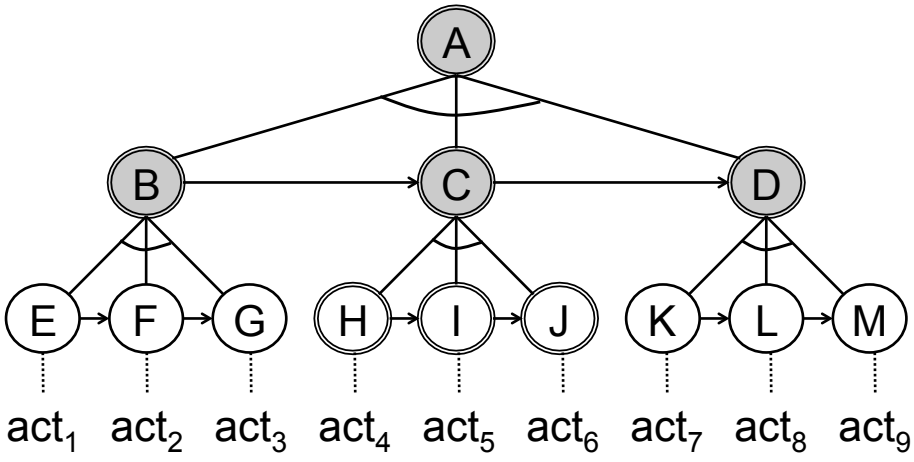

Figure 3. A Hierarchical plan structure with highlighting, Grey nodes are all part of a single HTN method, while nodes with a ringed edge at part of a single CCG category.

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. CCGs can be used to encode hierarchical plan structures like that of Hierarchical Task Networks(HTNs)((Erol, Hendler, & Nau, 1994; Ghallab, Nau, & Traverso, 2004). Consider the hierarchical plan structure shown in Figure 3. If we use HTNs to represent such a plan, each decomposition step is captured in a *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 order. 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 3. 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. Each method 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 in Figure 3. As such a single CCG category will generally be larger and contains more causal structure than an HTN method and can be thought of as being grounded in a single executable action, namely its anchor.

We can also alter our thinking about what CCG categories represent. Rather than thinking of a CCG category as only the syntax for recognition of a plan, we can view them as the semantics for the generation of a plan. Adding this change in viewpoint can provide us some very interesting 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 a traditional HTN method. Effectively, knowledge about the plan is being associated with the basic actions that are chosen as anchors to achieve high level plans. This is in strong contrast to HTN planning systems that have to account for the status, organization, and learning of *methods* that are not immediately grounded in individual actions. Instead, in a CCG grammar, all of the knowledge required for reasoning about the domain is directly linked to the executable actions in that domain.

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 requires 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 abstraction 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, a CCG category for this plan 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. 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. To address this HTNs use heuristics or fixed rules 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. In contrast, in a lexicalized approach each category is allowed to encode a different order for the expansions of its arguments. This allows the categories within the lexicon to make

different choices about the order of argument expansion making this approach much more flexible and tailorable to the domain.

4.1 The Planning Algorithm

This work proposes a new planner that 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 control when plans to achieve the subcategories are built. Thus, plans for arguments will be built in the order they occur in the category. Consider the following simple plan lexicon.

CCG: 4.

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

This is one of the possible lexicons for the plan structure shown in Figure 3 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 occur 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 4. This algorithm is 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. While the planning process is total order, it calls for adding actions both to the left and the right of the initial action placed in the plan. It is therefore similar to causal order planing that was advocated in early planning research(Penberthy & Weld, 1992).

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
  END-for
  RETURN P; }

Figure 4. High level recursive algorithm for plan generation.

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.

4.2 An Example

Suppose we want to generate a plan to achieve category A. Following the 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 added to 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 that 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 fixed heuristic domain specific information. It is information that is specific to the category itself. It is inherent in the lexicon, and can therefore be tailored to each individual plan in the lexicon.

4.3 Discussion of Use

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 from every state.

Thus, after a plan is constructed, the plan must be simulated to verify that it achieves the given category. To do this, like other planning systems, we add to each basic action's definition a set of precondition, post condition rules that specify the outcome of executing the action in different world states. We will call these rules *projection rules*.

Note, projection rules do not contain what are traditionally considered *applicability conditions*(Ghallab, Nau, & Traverso, 2004) that define when the action should be executed or other conditions that are designed to control the search for a plan. They only simulate the results of performing the action. Thus, we have placed all information about how to build a plan into the grammar's categories and information about what happens when actions are performed into the action's projection rules.

Once a plan is found whose simulation achieves the desired goal, it is returned to the user. If the simulation doesn't achieve the goal the search must backtrack and look for another plan. Thus, the algorithm in Figure 4 is implemented with a backtracking search to find an alternative plans if earlier ones fail to achieve the objective.

Thus, like Fast Forward (FF) (Hoffmann & 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. Therefore, 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 using the projection rules, 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 guid-

ance. 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 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.

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 outweigh even moderate decreases in performance relative to the state of the art.

5. Future Directions

We have only described the initial implementation of a planner based on lexicalized reasoning. As a result there are a number of areas that can be pursued in future work. For example, the incompleteness of the categories and lexicon, and the existence of projection rules and satisfaction conditions for all atomic categories suggests possible additional tests that could be done in the planning process. Since argument categories must be satisfied before the execution of the action, for the action to achieve its root result, We could imagine adding two tests to the building of plans. First, never build a plan for a leftward argument such that its satisfaction conditions are satisfied in the initial state. Second, as a plan is build for each argument, test that the satisfaction conditions for the argument are in fact achieved by the plan. These tests would allow us to cut short the plan generation process. In the case of test one, we can imagine not building part of the plan because the desired argument's satisfaction conditions are already met in the initial state. In the case of test two, we can terminate plan construction early because the plan built for the argument had failed and therefore the overall plan will fail as well.

However, it is worth recognizing that to add either of these tests to the plan generation process will require something akin to the *downward refinement property*(Bacchus & Yang, 1993) for hierarchical planning. Effectively both of these tests have to assume that the reason the category results in correctly built plans is that the subplans for each of the arguments are causally isolated. That is the plans for each of the arguments do not causally interact in any way. They cannot be allowed to produce a condition needed by a later argument's plan or modify any of the conditions of the initial state that are needed for a later argument's plan. If either of these conditions held, then simulation of the subplan

might erroneously fail because some precondition of the subplan had not been yet been achieved (ie. an earlier argument's subplan that achieved a predicate was not yet part of the plan, or had failed to correctly modify the predicate from the initial state). We believe this is too strong a requirement, and thus have not included such tests in the planner. We leave their inclusion as an area for possible future work if runtimes are too high for real world deployment.

Beyond more extensive testing and exploring methods for pruning plan search, early there are multiple possible directions for future work in the larger lexicalized reasoning research program. First, the argument for reasoning about actions using lexicalized representations would be significantly strengthened if the exact same parsing and generation algorithms we have outlined here for plan grammars were demonstrated to work for natural language grammars and tasks. As we have already alluded, the parsing algorithm presented here is not one taken from NLP. Demonstrating that it could be successfully used to parse natural languages and that our generation algorithm could be used to produce sentences would significantly strengthen our argument for lexicalized reasoning. Keep in mind, the plan recognition algorithm was extended to deal with multiple concurrent interleaved plans (which doesn't occur in natural language), however, the plan recognizer does work effectively in the single plan case. This gives us good reason to suppose that it will work for natural languages, but this has yet to be demonstrated.

Second, our argument for lexicalized reasoning about actions would be significantly strengthened by an account of how such grammars can be learned. Encouragingly, there is a significant body of current work on learning CCG grammars for NLP that may be effective in learning CCG action grammars(Clark & Curran, 2004; Thomforde & Steedman, 2011; Kwiatkowski et al., 2012). Related to this, we note that this formulation has separated knowledge about how to build plans from knowledge of the implications of executing actions in particular states. As such, these two different kinds of knowledge could be learned separately.

Finally, using the same grammars to both recognize and generate sequences of actions provides another possible benefit to be explored in learning by demonstration. If the same grammars are used for recognition and generation, then any plan grammar learned by observation immediately becomes a plan that can be executed. No new knowledge is needed, and the learned knowledge does not need to be translated into another form.

6. Conclusions

As we have argued, the link between planning and language has a long tradition. Further the use of formal grammars to drive both recognition and generation problems has a long tradition in NLP, but a much shorter history in reasoning about actions. We have argued that taking seriously the idea of using formal lexicalized plan grammars to drive both plan recognition and generation could provide significant leverage on both problems and provide a unified view of both language and planning phenomena.

In this paper we have outlined how CCGs, a lexicalized grammar formalism taken from NLP, can be used in exactly this way. This paper has formalized CCG plan grammars and shown how a planner can be built that uses the same lexicon as our previously developed plan recognizer. Further this paper has outlined some of the areas for future work and suggested why we think this area of research is promising. In the end this paper provides an important step toward a single shared understanding of both recognition and generation of both physical actions and language use.

References

Bacchus, F., & Yang, Q. (1993). Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, *71*, 43–100.

Baldridge, J. (2002). *Lexically specified derivational control in Combinatory Categorial Grammar*. Doctoral dissertation, University of Edinburgh.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, *41*, 164–171.

Carberry, S. (1990). *Plan recognition in natural language dialogue*. MIT Press.

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

Curry, H. (1977). *Foundations of mathematical logic*. Dover Publications Inc.

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

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

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

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory and practice*. Morgan Kaufmann.

Hockenmaier, J. (2003). *Data and models for statistical parsing with combinatory catagorial grammar*. Doctoral dissertation, University of Edinburgh.

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

Joshi, A., & Schabes, Y. (1997). Tree-adjoining grammars. *Handbook of Formal Languages, Vol. 3* (pp. 69–124). Springer Verlag.

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

Lashley, K. S. (1951). *The problem of serial order in behavior.* Bobbs-Merrill.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior.* Henry Holt.

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

Penberthy, S. J., & Weld, D. S. (1992). Ucpop: A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich, & W. Swartout (Eds.), *Kr'92. principles of knowledge representation and reasoning: Proceedings of the third international conference*, 103–114. San Mateo, California: Morgan Kaufmann.

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

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

Schabes, Y. (1990). *Mathematical and computational aspects of lexicalized grammars.* Doctoral dissertation, University of Pennsylvania.

Steedman, M. (2000). *The syntactic process.* MIT Press.

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

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