Advances in Cognitive Systems 4 (2016) 187-206 Submitted 8/2015; published 06/2016

Lexicalized Reasoning about Actions

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 how the same grammars can be used to direct planning. It does this as part of a larger program to establish and leverage linkages between reasoning about action and language understanding.

1. Introduction

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) has been mentioned (Pynadath & Wellman, 2000; Vilain, 1990). A relationship between natural language sentence generation and planning has also been discussed (Carberry, 1990; Petrick & Foster, 2013). However, while researchers often pay lip service to these relationships, their research programs are rarely informed by the results and methods of other programs. This paper presents some initial results from a research effort that attempts to take these interrelations more 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 has used the same knowledge structures to both build plans and recognize the plans of others. This is not to suggest that the same approaches have not been used in past work. Multiple representations, including hidden markov models (HMM) (Baum, Petrie, Soules, & Weiss, 1970) and hierarchical task networks (HTN) (Tate, 1977; Erol, Hendler, & Nau, 1994), have been proposed as representations for both planning and plan recognition. However, we are not aware of any work that has made the stronger commitment to use the same HMM or HTN for both tasks. If our eventual objective is to build systems that integrate multiple subsystems, there is much to be learned by requiring the sharing of structures between the tasks.

This paper presents 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 are based on a formal grammar model taken from current research on natural language processing (NLP). Specifically, this work will build on prior work (Geib & Goldman, 2011) using Steedman's Combinatory Categorial Grammars (CCG) (2000) for plan recognition, which has shown success producing state of the art run times while addressing a number of open issues from earlier work.

2. Motivation for using Grammars to Represent Plans

Past work has made arguments about the expressiveness of planning formalisms based on their relation to regular and context-free grammars (Erol et al., 1994). This paper goes further and argues for the explicit use of a grammar to capture knowledge of the ordering of actions that is needed for planning and plan recognition. This may be intuitively plausible for some, but an argument for this move is required. While the use of formal grammars has been very productive in natural language research, some might argue that it is not clear that action domains have the same kinds of structure that is captured in formal grammars. For example, in language, number and tense agreement are easily solved as syntactic phenomena (Pereira & Shieber, 1987) as part of the grammar, but these do not have obvious correlates in the action domain, and the linguistic separation between syntax and semantics is not as obvious in action settings.

We would argue that addressing issues of number and tense in language are not the primary use for grammar. First and foremost, such grammars are used to define the acceptable orderings for the words in a sentence. That is, they specify the set of well-formed sentences within the language. Defining the acceptable sequences of actions that might form a plan is an important part of plan recognition and planning knowledge, and it is very similar to the problem of defining the set of syntactically well-formed sentences in human language. Thus, while some details of the tasks being performed differ between action domains and language, the need in both for the underlying functionality is clear. We argue that leverage may be found by viewing them through a common lens. If we are interested in finding unifying frameworks for multiple cognitive tasks, then we must first make sure that the major features of the two problems can be brought into correspondance and then look for lessons to be learned from the possibility of more fine-grained alignment.

2.1 Why Not Multiple Grammars?

Even accepting the premise that a plan grammar might be useful for these tasks, some might argue that we should construct different grammars for the two tasks. For recognition, the grammer is designed to address *ambiguity*, that is determining the roles of tokens in a sequence when each can have multiple roles. In the case of generation, the grammar addresses *synonymy*, that is, choosing between different realizations of a goal. Thus, one might argue, each problem requires a different grammar.

However, this argument is really only about efficiency, as it already concedes that at least one grammar is needed. There are strong reasons to suppose that the yield of both grammars must be the same in a given domain. If this is not true, an agent might be found in the odd position of being able

to build and execute a plan while being unable to recognize the same plan when executed by others. Given this, and that the only role of the grammar is to define the sequences of well-formed tokens, then any meaningful differences between grammars for recognition and generation can only concern their efficiency at directing plan recognition or planning, not in the sets of acceptable sequences. However, we have already stated that the objective is to show that it is possible to use the same grammar for both planning and plan recognition. Arguments about efficiency of the grammar are interesting, but only once we have shown that the same grammar can be used for both tasks.

2.2 Contribution and Paper Organization

Thus, while the major contribution of this paper will be the presentation of a new planning algorithm, the central contribution for cognitive systems should be seen in the much larger canvas of lexicalized reasoning research. This research program's central objective is to produce methods for generation and recognition that operate on a common grammar framework, such that varying only the input grammar will solve of four different "AI hard" problems, namely language parsing, language generation, plan recognition, and plan generation. We believe that lexicalized reasoning can successfully address these four problems with a common representational framework. Note that such a 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 explores this idea. First, it provides formal definitions for CCGs, shows how they can represent plans, and briefly sketches of how previous work has used them to perform plan recognition. Next it discusses new work that details how CCGs can be used to direct plan generation. After this, it contrasts this new approach with existing HTN planning methods. The paper closes by discussing implications of the approach and outlining directions for future work.

3. Using CCGs to Represent Plans

This section lays out formal definitions for all of the notations and terms required for the rest of this paper. These definitions are illustrated with a running example from a simplified robotic domain based on the EU FP7 Xperience project. As such, it is focuses on "pick and place" operations to move objects from one location to another. In an effort to bridge terminological differences between the research areas of NLP and reasoning about actions, these definitions differ slightly from those presented in CCG work on NLP and even those used in our previous work (Geib, 2009) .

3.1 Category Definitions

CCGs define acceptable orderings of tokens using functional categories. They use two binary operators, forward slash ($/$) and backward slash ($\backslash$), to define functions from a set of *argument categories* to a *result category*. For both operators, the category on the left of the slash is the *result* and the set of categories on the right are the *arguments*. More formally, we have

Definition 3.1. *A set of* ***categories*** $\mathcal{C}$ *is defined recursively:*

Atomic categories: *A finite set of base categories denoted* $\{A, B, C, ...\} \in \mathcal{C}$.

Complex categories: *Given a set of categories $\mathcal{C}$, where $Z \in \mathcal{C}$ and $\{W, X, Y, ...\} \neq \emptyset$ and $\{W, X, Y, ...\} \in \mathcal{C}$, then $Z/\{W, X, Y, ...\} \in \mathcal{C}$ and $Z\backslash\{W, X, Y, ...\} \in \mathcal{C}$.*

The slash indicates where the function looks for its arguments, which must occur *after* the category for forward slash and *before* it for backslash. In addition, each atomic category has a single *satisfaction condition*.

Definition 3.2. *An atomic category's **satisfaction condition** is a (possibly empty) first order logical formula using only conjunction and negation of predicates.*

Intuitively, a satisfaction condition (SC) defines a set of states in the action domain. For example, consider a SC for the category H-FULL:

$$\text{H-FULL}(x_1) \ := \ [\ \textit{in-hand}(x_1) \wedge \ !\textit{on-table}(x_1)\ ].$$

Note that, in order to move beyond a strictly propositional representation, atomic categories may have variables (x_1 in this case) that range over objects in the domain. Bindings of such variables will also have scope over the SC assigned to the category. Unbound variables in the SC are assumed to be existentially quantified. Thus, given the assignment above, the ground atomic category instance H-FULL(cup23) would denote the set of all states of the world in which both *in-hand*(cup23) is true and *on-table*(cup23) is not.

For clarity, the rest of this paper adheres to the typographical conventions established here. Atomic categories are in capitals and logical domain predicates in italic lower case, possibly hyphenated. Single, lower-case, italic letters with subscripts represent domain object variables, and domain object identifiers are identified by a lower-case object name followed by a number. Action identifiers are in bold lower case.

Assigning SCs to atomic categories lets us view a category as a function from states to states. To see this, consider atomic categories. With the assignment of SCs, each atomic category can be seen as a zero arity function that maps from any possible state of the domain to one of the states defined by the category's SC. For example, assume we assign the action **grasp** the atomic category H-FULL:

$$\textbf{grasp}(x_1) \ := \ [\ \text{H-FULL}(x_1)\ ].$$

This tells us that the ground action instance **grasp**(cup23) is a function such that executing it in any state results in being in a state where *in-hand*(cup23) is true and *on-table*(cup23) is not. Complex categories define more restricted functions and introduce notions of abstraction and seriation.

3.2 Complex Categories Using Backward Slash

Remember that the backslash operator ($\backslash$) defines a function whose argument categories must occur before it in order to achieve the states defined by the satisfaction condition of the resulting category. Consider adding another atomic category (H-AROUND) to our domain and changing the definition of **grasp** to

H-AROUND(x_1) := [*hand-around*(x_1)]
grasp(x_1) := [H-FULL(x_1) \ { H-AROUND(x_1) }]

so that **grasp** is a function that results in the SC of H-FULL being true, but only if, immediately before its execution, another function is executed whose result category is H-AROUND. Alternatively, **grasp** is a function that achieves the SCs of H-FULL, but only if it is executed in states where the SCs of H-AROUND have already been satisfied.

It is worth noting that limiting categories to a single leftward slash lets CCGs represent STRIPS-style actions (Fikes & Nilsson, 1971) with the form

EFFECTS \ { PRECONDITION }

where the SC of EFFECTS encodes the effects of the action and the SC of PRECONDITION specifies the preconditions of the action. However, given the functional nature of the representation, if another argument is added to the category, this parallel no longer holds. Consider these definitions

H-FULL(x_1) := [*in-hand*(x_1) ∧ !*on-table*(x_1)]
H-AROUND(x_1) := [*hand-around*(x_1)]
H-EMPTY := [!*in-hand*(x_1)]
grasp(x_1) := [(H-FULL(x_1) \ { H-EMPTY}) \ { H-AROUND(x_1) }]

Here **grasp** is a function that achieves H-FULL, but only if a function is executed before it that results in H-AROUND and only if a still earlier function is executed that achieves H-EMPTY. While this is a natural extension of the CCG formalization, such information cannot be explicitly expressed in STRIPS-style action representations. Such ordering information is much closer in style to an HTN representation. Section 5 will say more about the relation of CCGs to HTNs.

3.3 Complex Categories Using Forward Slash

The forward slash operator works in the same way as the backslash but expects its arguments to occur later. Consider a set of definitions for putting an object on the table:

H-FULL(x_1) := [*in-hand*(x_1) ∧ !*on-table*(x_1)]
H-EMPTY := [!*in-hand*(x_1)]
ON-TABLE(x_1) : [*on-table*(x_1)]
reach4pl(x_1) := [(ON-TABLE(x_1) / { H-EMPTY}) \ { H-FULL(x_1) }]
release := [H-EMPTY]

In this case, we could undestand the ground action instance **reach4pl**(cup23) as a function that results in *on-table*(cup23) being true if, prior to its execution, the SCs of H-FULL(cup23) are true and after its execution an action that has H-EMPTY as its atomic result is executed. Expressing such forward relationships is impossible in STRIPS-style action representations.

Thus, CCG categories define functions from states to states, with atomic categories defining functions that map from any state to a specific state, and with complex categories being more restrictive. In the end, the two slash operators let us build up *curried functions* (Curry, 1977) from

states to states. We allow multiple categories to be assigned to an action to define alternative possible functional mappings for a single action.

3.4 Projection Rules

Section 4 discusses how such categories can be used to direct planning. To do this it is important to have detailed *projection rules* that predict the effects of executing an individual action within a given state. As mentioned, such knowledge could be represented using CCG categories, but, to simplify our exposition, we use better known precondition and postcondition notation. For example, a partial set of rules that could be associated with the actions **release**, **reach4gr**, **reach4pl**, **grasp**, and **unreach** is

release :: [*in-hand*(x_1)] $\rightarrow$ [!*in-hand*(x_1) $\wedge$ *hand-empty*]
reach4gr(x_1) :: [*hand-empty* $\wedge$ *hand-at-side*] $\rightarrow$ [*hand-around*(x_1) $\wedge$!*hand-at-side*]
reach4pl(x_1) :: [!*hand-empty*] $\rightarrow$ [!*hand-at-side*]
grasp(x_1) ::
[*hand-around*(x_1) $\wedge$ *hand-empty*] $\rightarrow$ [*in-hand*(x_1) $\wedge$!*hand-empty*$\wedge$!*hand-around*(x_1)]
unreach :: [!*hand-at-side*$\wedge$ *in-hand*(x_1) $\wedge$ *on-table*(x_1)] $\rightarrow$ [*hand-at-side*$\wedge$!*on-table*(x_1)]

where each rule has the form action :: precondition $\rightarrow$ postcondition. Such projection rules do not contain what have been called *applicability conditions* (Ghallab, Nau, & Traverso, 2004) that specify when the action should be executed or other conditions that control the search for a plan. They only simulate the results of performing the action. For example, such a rule lets one infer that, if **release** is executed in a state where *in-hand*(cup23) is true, it results in a state where *in-hand*(cup23) is not true and *hand-empty* is true. To simulate the execution of an action in a state, only a single rule is applied. The precondition of the rule is matched using negation by failure. If no rule has a condition that matches, then there is no knowledge about the action's effects and it is assumed to have no effect. Thus, if **grasp**(cup23) is executed in a state where *hand-around*(cup23) is NOT true, then (assuming the set of rules above is complete) one should conclude that the action has no effect on the domain. Section 4 discusses the use of such rules.

3.5 Plan Lexicons

Having placed information about how to build a plan into the grammar's categories and information about action execution into projection rules, we are now in a position to define a CCG plan lexicon.

Definition 3.3. *A **plan lexicon** is a tuple $PL = \langle \Sigma, \mathcal{C}, f \rangle$, where Σ is a finite set of 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 to which an observed action type σ can be assigned. For brevity, one often just provides the function that maps observable action types to categories to define a plan lexicon. For example, CCG:1 provides a partial lexicon for the reaching domain

release	**reach4gr**(cup23)	**grasp**(cup23)	
H-EMPTY	H-AROUND(cup23)	((PICK(cup23)/{AT-REST})\{H-EMPTY})\{H-AROUND(cup23)}	
		(PICK(cup23)/{AT-REST})\{H-EMPTY}	<
		PICK(cup23)/{AT-REST}	<

Figure 1. Parsing observations with combinatory categorial grammars.

CCG: 1.

release := [H-EMPTY],
reach4gr(x_1) := [H-AROUND(x_1)],
grasp(x_1) := [((PICK(x_1) / {AT-REST}) \ {H-EMPTY}) \ {H-AROUND(x_1)}],
unreach := [AT-REST].

Although only a single category is associated with each action in this example, this is atypical and a result of the brevity of the example.

Two other terms will also be helpful in the remainder of the paper.

Definition 3.4. *A category R is 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).*

For example, PICK is the root result of the category

((PICK(x_1) / {AT-REST}) \ { H-EMPTY}) \ { H-AROUND(x_1) }.

Definition 3.5. *An action type **a** is a possible **anchor** of a plan for category C if the lexicon assigns to **a** at least one category whose root result is C.*

For instance, in CCG:1, **grasp** is the anchor for PICK. This formulation of CCGs is closely related that of Baldridge (2002) in allowing sets of arguments for categories. Sets of arguments are critical for the treatment of partial ordering in plans. The interested reader can find details of this idea in Geib (2009) and in Geib and Goldman 2011.

3.6 CCG Combinators

Next, we must show how CCG categories are combined into higher-level plan structures. We use, three binary *combinators* (Curry, 1977) to combine the categories of individual observations:

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

Here X and Y are categories, while α and β are possibly empty sets of categories. To see how a lexicon and combinators parse observations into high-level plans, consider the derivation in Figure 1

for the sequence of observed actions **release**, **reach4gr**(cup23), **grasp**(cup23). As each observation is encountered, it is assigned a category on the basis of CCG:1 and then combinators combine the categories. For example, first **release** is observed and assigned to H-EMPTY but no combinators can be applied. Next **reach4gr** is assigned to H-AROUND; again, no combinators are applicable. Once **grasp** is observed and assigned its category, leftward application is used twice to combine both the H-AROUND and H-EMPTY categories with **grasp**'s initial category, resulting in PICK(cup23) /{AT-REST}. The use of the combinators requires the unification of variables in the actions or categories. In this case, if the observed action had been **reach4gr**(cup46), the leftward application combinator would not have applied, because cup46 and cup23 do not unify.

3.7 Plan Recognition Using CCGs

Given a set of *observations* and a CCG lexicon defining a set of possible plans, we can view plan recognition as a kind of parsing. Geib (2009) and Geib and Goldman 2011 have shown that, at a high level, this involves three steps:

1. Build the complete and covering set of parses (or *explanations*) that organize the observations into one or more plan structures that meet the requirements defined in the plan lexicon, as seen in Figure 1.

2. Establish a probability distribution over the explanations.

3. Since there can be a large number of explanations and each can contain multiple hypothesized plans, compute the conditional likelihood of each root-result from the distribution over the explanations.

The ELEXIR plan recognition system implements this approach, and we have shown that it addresses a number of outstanding problems in plan recognition, including multiple interleaved plans, plans with loops, and partially ordered plans, all with efficient run times. However, the central contribution of this paper is using the same representations for generation. Thus, a complete discussion of plan recognition is outside its scope. We refer the interested reader to Geib (2009) and to Geib and Goldman 2011 for a fuller discussion of ELEXIR.

4. Using CCGs for Planning

Having presented an overview of CCGs and their use in plan recognition, we can now look at using these same grammars to direct plan construction. Intuitively, the same knowledge of action orderings used in plan recognition can direct the building of a plan.

A cognitive system can use the directionality and order of the arguments in a CCG category not only to capture ordering relations among argument categories, but also to control the construction of a plan to achieve the categories' root result. That is, the structure of a category directly reflects both the ordering constraints on the plan's substeps, as well as the order of the search for a plan to achieve the root result. Thus, sub-plans to achieve each argument of the category are built in the order they occur within the category.

Table 1. High-level recursive psudeocode for plan generation. Backtracking in this search both over category choices and category variable bindings has been omitted for simplicity of exposition.

Procedure $BuildPlan(G)$ {
  LET $\mathcal{C}_G$ = the set of all $c_i \in \mathcal{C}$ such that $G = root(c_i)$;
  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; }

Taking this view, using CCGs to plan requires two operations. First, to build a plan for a category, the system must choose an action that is one of its anchors. This is the inverse of assigning a category to an observed action during plan recognition. This step binds any variables associated with the category. Second, if the chosen action's category is complex, the system recursively builds plans for each of the category's arguments. The system appends the resulting sub-plans either to the left or right as determined by the category's slashes. This corresponds to the application of the CCG combinators in parsing. Table 1 gives pseudocode for this procedure. To make the logic of the category directed search as clear as possible, it does not include code for binding and searching over individual category variables. In Table 1 $\mathcal{C}$ denotes the set of all categories and $\overrightarrow{\mathbf{v_x}}$ is a variable over category structures.

Keep in mind that the order of a complex category's arguments determines the order in which the planning process occurs, and the direction of the slash determines where the subplan is added to the plan. Although the resulting plan is totally ordered, actions can be added both to the left and right of the first action placed in the plan. In this, it is similar to causal-order planing (Penberthy & Weld, 1992). It is also worth noting that, the definition of complex categories does not restrict which categories can be arguments. This means that complex categories can be recursive and therefore can define plans of arbitrary depth. CCGs of the type we used here are known to have the same expressiveness as context-free grammars (Joshi, Shanker, & Weir, 1990) and are thus in the same expressiveness class as Hierarchical Task Network (HTN) planners.

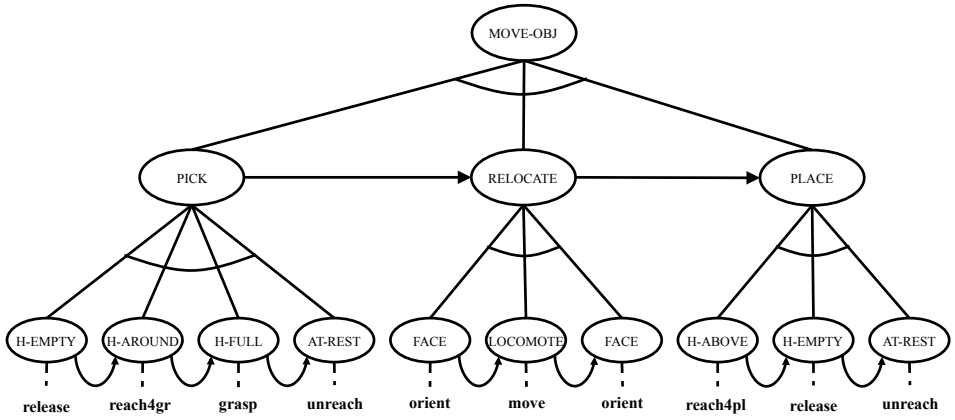

Figure 2. A hierarchical partially ordered plan to relocate an object.

4.1 An Example

An example will aid in understanding how such plans are built. Consider a simple plan lexicon fragment:

CCG: 2.

PICK(x_1) := [*in-hand*(x_1)],
H-FULL(x_1) := [*in-hand*(x_1) $\wedge$!*on-table*(x_1)],
H-EMPTY := [!*in-hand*(x_1)],
...
release := [H-EMPTY, (PLACE(x_1)/{AT-REST})\{H-ABOVE(x_1)}],
reach4gr(x_1) := [H-AROUND(x_1)],
reach4pl(x_1) := [H-ABOVE(x_1)],
unreach := [AT-REST],
grasp(x_1) := [((PICK(x_1)/{AT-REST})\{H-EMPTY})\{H-AROUND(x_1)}],
orient(x_1) := [FACE(x_1)],
move(x_1) :=
[(((MOVE-OBJ(x_1, x_2)/{PLACE(x_2)})/{FACE(x_1)})\{PICK(x_2)})\{FACE(x_1)}].

This is part of a lexicon used in the hierarchical plan structure in Figure 2. It extends CCG:1 to build plans for moving objects from one location to another. While all of the actions for the domain are shown, we have omitted some satisfaction conditions and projection rules. It is also worth noting that action **release** has two categories, one that only empties the hand and another that places an object successfully on a surface.

Suppose our goal is to build a plan to achieve the state *in-hand*(cup23). To start the system finds satisfaction conditions for the atomic categories that cover the desired state. If this process identifies PICK as having an appropriate satisfaction condition, then it unifies the SC and goal state to produce bindings for the category's variables. In this case, this results in PICK(cup23). Using typography

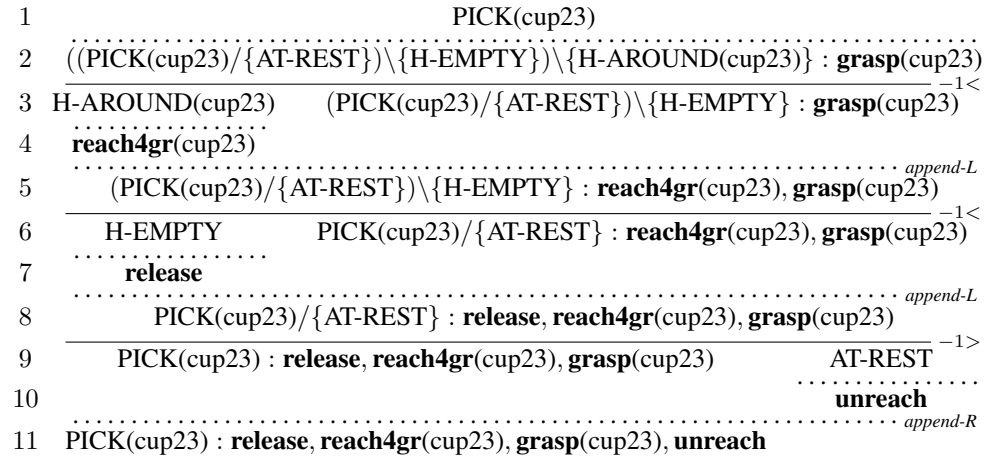

Figure 3. Building a plan to achieve the goal category PICK(cup23). Solid lines denote the action of deconstructing a complex category (-1 and a direction indicator). Dotted lines separate two subtasks of the planning process: choosing an action to achieve a particular category (no annotation) and adding the chosen action to the plan ("append-L" or "append-R") .

similar to that in the earlier parsing diagrams, Figure 3 shows how this instantiated category directs the search for a plan.

First, given a category, the system selects an anchor action that has the category as its root result and binds the action's variables on the basis of the category's bound variables. For example, on the second line of Figure 3, it has chosen the action **grasp** as the anchor for PICK, built a bound instance of the action, **grasp**(cup23), and created a grounded instance of the category to direct the rest of the search for a plan:

((PICK(cup23)/{AT-REST})\{H-EMPTY})\{H-AROUND(cup23)}

Next the system adds the chosen action to the plan. Since this is the first action to be added, no interesting reasoning is required. The system then looks at the next argument of the current category, in this case H-AROUND(cup23) (see line three of Figure 3) and repeats the process. In line 4, the system has selected an action that is an anchor for H-AROUNDand bound it, resulting in

reach4gr(cup23)

Since H-AROUNDit only requires a single action, planning for this subcategory is now complete. In line 5, because H-AROUND was a leftward argument of the category directing the plan search, the system adds the action to the front of the current plan, resulting in the plan fragment:

P = [**reach4gr**(cup23), **grasp**(cup23)]

In lines 6 to 8, the same process is repeated on the H-EMPTY category, giving the plan fragment:

P = [**release**, **reach4gr**(cup23), **grasp**(cup23)]

Lines 9 to 11 in the figure see the same process repeated on the AT-REST category, with the significant difference that in this case a rightward-looking argument to the category directs the search. As a result, line 11 adds the **unreach** action to the end of the current plan:

P = [**release**, **reach4gr**(cup23), **grasp**(cup23), **unreach**]

Because the category directing search has no more argument categories, the plan is complete. Again, the actions were not added strictly from left to right. Instead, as each argument's subplan was completed, it was added either to the beginning or end as dictated by the category. Thus, the plans is built from the middle outward.

4.2 Verifying the Plan

The plan lexicon encodes methods for building plans that are *likely* to succeed. They need not possess the *downward refinement property* (Bacchus & Yang, 1991) and, as such, are not guaranteed to work. Rather, each CCG category represents a suggestion about how to go about building a plan that might achieve the desired goal. This contrasts with most HTN planners, which assume the specified set of plans is complete. In this style of planning, the search for a plan to achieve a particular goal category is informed by the categories of the CCG lexicon, but the lexicon is not guaranteed to contain a successful plan from every state.

Thus, after construction, the system simulates the plan to verify that it achieves the goal. To do this, it employs the projection rules defined for each action in the lexicon that we described in Section 3.4. In our example, if one knows that the initial state of the world for the plan is

[*hand-empty*$\wedge$ *hand-at-side*$\wedge$ *on-table*(cup23)]

then, by following the plan and using the projection rules from Section 3.4, the system can infer that the generated plan will produce the state

[*in-hand*(cup23) $\wedge$ *hand-at-side*]

If a plan's simulation results in a desired goal, the system returns it to the user. If the simulation does not achieve the goal, then search backtracks and looks for another plan. Thus, like FastForward (Hoffmann & Nebel, 2001) and other modern planners, this method is incomplete. The lexicon encodes heuristic knowledge about likely ways to achieve goals. Therefore, if completeness is required and if the lexicon does not support a plan, one must fall back on search through the space of all basic action sequences using the projection rules.

We have built a full, first-order logical planner in C++ that implements these ideas using the same core data structures and representations as the ELEXIR recognizer. We believe that the use of common underlying data structure and grammar formalism for both planning and plan recognition is an intellectual contribution that outweighs even moderate decreases in performance relative to other systems.

5. Relation to Hierarchical Task Networks

The action representation most closely related to combinatorial categorial grammars (CCGs) is clearly hierarchical task networks (HTNs) (Tate, 1977; Ghallab et al., 2004). Both support hierarchical structure, ordering constraints, and capture domain knowledge about how to build or recognize plans that goes beyond non-hierarchical approaches. Thus, it is worth spending some time to understand their differences. There are multiple formulations of HTNs that differ in details. Georgievski and Aiello 2014 offer a relatively complete discussion of these alternative formulations. Here we compare CCGs to the major features that are shared by most HTN formalisms.

5.1 Background

Consider the hierarchical plan structure for transferring a single object from a sideboard to a table shown in Figure 2. HTNs have a set of *basic actions* of the kind found in non-hierarchical planners. Following our previous typography, we denote these actions with lowercase bold identifiers. This representation of these is very similar to that found in STRIPS (Fikes & Nilsson, 1971) and other non-hierarchical planing systems. As such, they have preconditions and effects that capture the logic necessary to project state changes caused by the action's execution. In addition, HTNs have *tasks* that are defined by *methods*, which specify the results of decomposing high-level tasks into component subtasks. To simplify our discussion, like CCG category identifiers, we will use capitalized identifiers for tasks (e.g., MOVE-OBJ) but we will shortly discuss their differences.

If we use HTNs to represent the plan structure shown in Figure 2, each of the non-basic actions requires a decomposition encoded by a method. A method captures the fact that, to achieve the high level task MOVE-OBJ, the subtasks PICK, RELOCATE, and PLACE must be executed in order. This would look something like

Method1: MOVE-OBJ→ PICK, RELOCATE, PLACE

and would only convey information about the top four tasks in Figure 2. For the purposes of discussion, we will use a propositional rather than first order representation, but nothing we will say hinges on this assumption. There might also be methods for decomposing PICK, PLACE, and RELOCATE:

Method2: PICK→ H-EMPTY, H-AROUND, H-FULL, AT-REST
Method3: RELOCATE→ FACE, LOCOMOTE, FACE
Method4: PLACE→ H-ABOVE, H-EMPTY, AT-REST

HTN's methods make no commitments about how the subtasks should be decomposed or the order of their decomposition. Each method only captures a single level of decomposition from high-level task to an ordered sequence of subtasks. In some implementations, a method must only provide a partial ordering over the subtasks in its definition. Furthermore, each method may have a set of preconditions that specify when it can be used, and some variants let methods specify effects (Georgievski & Aiello, 2014).

Finally, HTNs have a set of methods that map tasks to basic actions which can be executed to achieve the task.

Method5: H-ABOVE→ **reach4pl**	Method6: H-AROUND→ **reach4gr**
Method7: H-FULL→ **grasp**	Method8: H-EMPTY→ **release**
Method9: AT-REST→ **unreach**	Method10: LOCOMOTE→ **move**
Method11: FACE→ **orient**	

In this example, each action maps to a single task, but this is not required. A task might map to a set of basic actions that achieve it or, depending on the treatment of HTNs, basic actions could occur as siblings of tasks in any other method definition.

There are a number of techniques for building plans using HTNs. The most common is relatively simple. An external mechanism (typically the user) chooses a highest level task, then recursive search expands and orders the task's subtasks to produce a sequence of basic actions. This search takes place over multiple decision points:

1. choosing a method chosen to expand a task;
2. choosing an ordering over partially ordered subtasks; and
3. choosing an order for the expansion of any subtasks.

Organizing this required search for HTNs is at the core of the first difference from CCGs we will highlight.

5.2 Lexicalization

Any lexicalized grammar has the benefit of isolating domain-dependent information in the lexicon and precompiling search. Consider again the HTNs that encode the plans in Figure 2. Method1 expands MOVE-OBJ, to a set of lower level tasks, but it does not add any basic actions to the plan. In general, substantial search and structure building may be required before adding any basic actions. In contrast, consider building the same plan based on CCG:2. The first step is to add the action **grasp** to the plan and, with this, the entire high-level structure of the plan. This results from the plan structure being encoded in the CCG lexicon.

Such precompilation of grammars is well known. Context free grammars for specifying programming languages are frequently rewritten into Greibach Normal Form (Greibach, 1965) precisely to force each production rule to have at least one non-terminal and thereby reduce search. The search that CCGs remove can also be removed from an HTN by lexicalizing the CFGs that implicitly define it. Requiring each HTN method to contain at least one basic action in the task expansion can effectively lexicalize HTN methods. For example, we can imagine a process very similar to that used to rewrite grammars to Greibach Normal Form that rewrites Method4 to contain a basic action with the form

Method4: PLACE→ H-ABOVE, **release**, AT-REST.

We could then rewrite this method as the CCG lexicon entry

release := (PLACE/{AT-REST})\{H-ABOVE}

This would transform the HTN into something that looks like a CCG, but that retains causal ordering information. IN this way, one can remove at least some of the search for a plan in the same way as CCGs. However, it requires assumptions about the ordering in which subplans are built and equating HTN tasks with CCG categories, which, as we will see next, are very different.

5.3 Foundations and Grounding HTNs

Much of the prior work on HTN planning has provided only procedural definitions for some of the core concepts, leaving open fundamental questions:

1. What is the formal nature of a task?
2. What is the relationship of the method's preconditions and effects to its task?
3. What is the relationship of the method's preconditions and effects to the basic actions that expand it?

We address these questions here. First, as we have already noted, categories are unlike tasks in that they are approximate functions from states to states. Thus, categories require no additional ontological machinery to specify complex categories. The representation of basic actions used by both CCGs and HTNs are approximate functions between states and categories capture this relationship.

Second, HTNs offer only a procedural understanding of a task's preconditions. Modern planning research distinguishes among applicability preconditions', enablement preconditions', and secondary preconditions'. In general, preconditions in HTNs describe applicability, in that they must be true in the current world state for the method to expand the task. As such, preconditions encode another way to control search beyond the method definition. Our definition of categories eliminates preconditions and thereby unifies search control knowledge into category structures. Thus, there are no conditions that must hold before one executes an action. In unusual contexts, an action's projection rules may be unable to predict the effects of an action, but CCG lexicons do not limit the states in which a plan can be recognized or an action executed. Instead, the categories alone constrain search for a plan.

Third, as Bacchus and Yang 1994 have discussed, without additional constraints in the form of the downward refinement property and assumptions about the relationship between abstraction layers of a hierarchical plan, there is no necessary relation between a task, a method, and the existence of a sequence of actions that achieve it. However, as we have presented them, each CCG category is anchored by a basic action and makes a commitment not present in HTNs. Each category definition requires that, if following the category results in a sequence of actions that achieve the success conditions of the root result, this sequence must include the anchor action. That is, choosing the complex category commits to the presence of at least one basic action within the plan.

Further, CCGs as defined here do not guarantee that every sequence of actions derivable from them will be an acceptable plan for the satisfaction conditions of the category's root result. They support a much weaker claim: categories provide heuristic guidance for the recognition or building

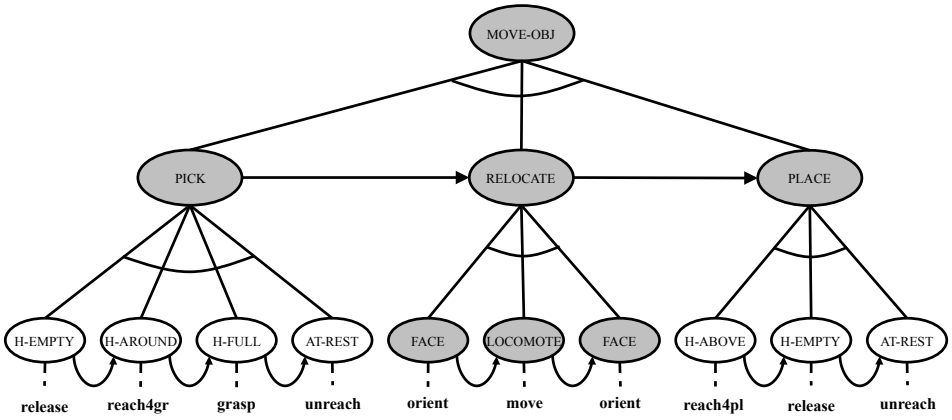

Figure 4. The hierarchical plan structure from Figure 2 indicating the coverage of a single CCG category from CCG:2. All shaded nodes are encoded in the complex category for the **move** action.

of plans. Their relationship to achieving satisfaction conditions of the categories root result is not causal or definitional. Instead categories capture the beliefs of the agent about likely correct methods for building plans.

To summarize these differences, we have provided a formal definition of CCG categories that goes beyond the procedural foundations of many HTNs planners. Further, CCG categories unify and generalize the search-control knowledge that is captured by multiple structures in HTNs, thus providing a simplified understanding of such knowledge. Finally, viewing CCG categories in terms of heuristic search control, rather than as abstract causal knowledge, separates knowledge about how to build plans from knowledge of physics of the world.

5.4 Structural Differences from HTNs

A single CCG category captures the spine of a tree from the root of the tree (or a subtree) to a basic action at a leaf. For example, in CCG:2 the action **move** is the anchor for a plan to achieve MOVE-OBJ. This requires the lexicon include a category of the form repeated here from CCG:2:

move(x_1) :=
[(((MOVE-OBJ(x_1, x_2)/{PLACE(x_2)})/{FACE(x_1)})\{PICK(x_2)})\{FACE(x_1)}]

Figure 4 shows a graphical representation of the plan defined by CCG:2 with the portions of the plan tree are covered by **move**'s category in grey. This single category combines the information in the HTN Method1 and Method3. Thus, the combined use of these two methods is captured in the category for the action that is its anchor. Lexicalization of the plan grammar places the knowledge about the use of basic actions in the categories for the actions themselves. This is why the category for **move** defines a substantially larger portion of the plan's causal structure than a traditional HTN method. Knowledge about the plan is associated with the basic actions that serve as anchors to

achieve high-level plans. This contrasts strongly with HTN planning systems, which must account for the status and organization of methods that are not immediately grounded in individual actions.

This category indicates that **move** is one of the actions *in the middle* of a plan to achieve MOVE-OBJ. Furthermore, it identifies the subcategories that must be achieved before and after it. However, some of the subcategories occur at different levels of abstraction in the plan space. For example, in most HTNs, PLACE and FACE would appear at different stages of the decomposition process. A CCG category for this plan contains more information about how to achieve MOVE-OBJ than the analogous HTN method and cuts through traditional HTN decompositional layers to an anchoring action. In this way, they are like lexicalized tree grammars (Joshi et al., 1990) and, as such, are generative and can produce plans of arbitrary depth. For example, any category with PLACE as its root result can guide both recognition and building of plans to achieve the categories' satisfaction conditions in any context. We are not constrained to only use the method shown in Figure 4. We could use any category that had PLACE as its root result.

The way in which CCGs are used also qualitatively changes the way in which one looks at the information in the lexicon. CCGs not only define the subtasks necessary to accomplish a task, but also the order in which they should be built. This is an important distinction. HTNs say nothing about the order in which the subtasks should be further decomposed. This additional domain knowledge is usually encoded in the HTN search engine itself. In contrast, in a lexicalized grammar, each category can encode the domain-specific knowledge of the expansion order for its arguments. Thus, a single lexicon can encode different domain-specific search choices for different categories and retain all of this knowledge in the lexicon.

To summarize this subsection, CCGs provide a single framework to understand preconditions, effects, causally prior tasks and causally subsequent tasks, all of which have all been treated separately in the HTN literature. As a result, CCGs, unlike HTNs, can define actions at any level of abstraction as a function from states to states. Further, much of the search for a plan is compiled into the CCG categories, capturing plan tree spines rather than an HTN method's single step of decomposition. Finally, domain-specific control knowledge about how to recognize or build a plan, which must be encoded separately in HTNs, is naturally part of the CCG categories.

6. Directions for Future Research

There are multiple directions for future work in the lexicalized reasoning research program. First, the argument for reasoning about actions using lexicalized representations would be strengthened if the same parsing and generation methods described here are shown to work for natural language grammars. Our parser for plan recognition, covered in previous work, is not taken from natural language research. The parser extends work in natural language process to address multiple, concurrent, interleaved plans, which do not occur in natural language, but the plan recognizer does work in the single plan case. This gives us good reason to believe that it will work for natural languages, but we must still demonstrate this ability.

Demonstrating the planner's capacity for generating sentences would also strengthen our argument for lexicalized reasoning. Prior work has shown that conventional planners can be used for this task (Koller & Petrick, 2011). There is no reason to suppose our planer would be ineffective for

this purpose, but demonstrating it would lend further support to the lexicalized reasoning research program.

Our argument for lexicalized reasoning about actions would also be strengthened by an account of learning plan grammars. Encouragingly, there is a considerable body of work on learning CCG language grammars that may be effective for learning the plan lexicons described here (Clark & Curran, 2004; Thomforde & Steedman, 2011; Kwiatkowski, Goldwater, Zettlemoyer, & Steedman, 2012). Using the same grammars to recognize and generate sequences of actions provides a particular advantage for learning by demonstration (Nejati, Langley, & Konik, 2006). Consider learning by observing another agent. Assuming that the correct plan can be recognized from a demonstration, then if different representations are used for recognition and generation, the identified plan must still be converted or translated to use it for generation. Still more learning may be required for this conversion. If the same grammar is used for both tasks, this eliminates the problem. The fragment of the grammar used to recognized plan can directly generate a new instance of the plan. No mapping between the representations is required and no additional learning needed. Thus, learning by demonstration may be particularly productive for such systems.

7. Conclusions

As we have argued, the link between planning and language has a long tradition. Moreover, the use of formal grammars to drive both recognition and generation has a long tradition in natural language processing, but a much shorter history for reasoning about actions. We argued that using lexicalized plan grammars to drive both plan recognition and generation could provide leverage on both problems and offer a unified view of language and planning.

In this paper, we outlined how CCGs, a lexicalized grammar formalism taken from natural language processing, can be used in this way. We specified CCG grammars for plans and showed how a planner can use the same lexicon as previously developed for a plan recognizer. We also outlined areas for future work and suggested why this line of research is promising. In summary, the paper presents an important step toward a shared understanding of recognition and generation for planning and language.

Acknowledgements

The research leading to these results was partially funded by the European Commission's Seventh Framework Programme under Grant No. 270273 (Xperience). The author would like to acknowledge Ron Petrick and Mark Steedman for helpful discussions and critical questions in the development of these ideas and Pat Langley and the anonymous reviewers from the 2015 Advances in Cognitive Systems Conference for insightful questions on the relation between this formalism and prior work and comments on earlier versions of this paper.

References

Bacchus, F., & Yang, Q. (1991). The downward refinement property. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 286–292). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Bacchus, F., & Yang, Q. (1994). 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*. Ph.D. thesis, University of Edinburgh, School of Informatics, Edinburgh, Scotland.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring 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*. Cambridge, MA: MIT Press.

Clark, S., & Curran, J. (2004). Parsing the WSJ using CCG and log-linear models. *Proceedings of the Forty Second Annual Meeting of the Association for Computational Linguistics* (pp. 104–111). Barcelona, Spain: Association for Computational Linguistics.

Curry, H. (1977). *Foundations of Mathematical Logic*. Mineola, NY, USA: Dover Publications Inc.

Erol, K., Hendler, J. A., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 1123–1128). Menlo Park, CA, USA: AAAI Press.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *3*, 189–208.

Geib, C., & Goldman, R. (2011). Recognizing plans with loops represented in a lexicalized grammar. *Proceedings of the twenty fifth Conference on Artificial Intelligence* (pp. 958–963). San Francisco, CA, USA: AAAI Press.

Geib, C. W. (2009). Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence* (pp. 1702–1707). Pasadena, CA, USA: AAAI Press.

Georgievski, I., & Aiello, M. (2014). An overview of hierarchical task network planning. *ArXiv e-prints*. http://adsabs.harvard.edu/abs/2014arXiv1403.7426G.

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

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

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Joshi, A. K., Shanker, K. V., & Weir, D. (1990). *The convergence of mildly context-sensitive grammar formalisms*. Technical Report MS-CIS-90-01, University of Pennsylvania, Department of Computer and Information Science, Philadelphia, PA, USA.

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

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. *Proceedings of the Thirteenth Conference of the European Chapter of the Association for Computational Linguistics* (pp. 234–244). Avignon, France: Association for Computational Linguistics.

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

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior*. New York, NY, USA: Henry Holt and Company.

Nejati, N., Langley, P., & Konik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the 23rd International Conference on Machine Learning* (pp. 665–672). New York, NY, USA: ACM. URL `http://doi.acm.org/10.1145/1143844.1143928`.

Penberthy, S. J., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR '92)* (pp. 103–114). Cambridge, MA, USA: Morgan Kaufmann Publishers Inc.

Pereira, F. C. N., & Shieber, S. M. (1987). *Prolog and natural-language analysis*. Stanford, CA, USA: Center for the Study of Language and Information.

Petrick, R. P. A., & Foster, M. E. (2013). Planning for social interaction in a robot bartender domain. *Proceedings of the Twenty Third International Conference on Automated Planning and Scheduling* (pp. 389–397). Rome, Italy: AAAI Press.

Pynadath, D., & Wellman, M. (2000). Probabilistic state-dependent grammars for plan recognition. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (pp. 507–514). Stanford, CA, USA: Morgan Kaufmann Publishers Inc.

Steedman, M. (2000). *The syntactic process*. Cambridge, MA, USA: MIT Press.

Tate, A. (1977). Generating project networks. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 888–893). Cambridge, MA, USA: Morgan Kaufmann Publishers Inc.

Thomforde, E., & Steedman, M. (2011). Semi-supervised CCG lexicon extension. *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (pp. 1246–1256). Edinburgh, UK: Association for Computational Linguistics.

Vilain, M. (1990). Getting serious about parsing plans. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 190–197). Boston, MA, USA: AAAI Press.