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

Deliverable Number:	D4.2.3
Deliverable Title :	Learning Plans for Plan Recognition Using Structural Bootstrapping With a Focus on Simple Plans
Type (Internal, Restricted, Public):	PU
Authors:	Christopher Geib
Contributing Partners:	UEDIN

Contractual Date of Delivery to the EC:	31-01-2014
Actual Date of Delivery to the EC:	01-04-2014

Contents

Chapter 1

Summary

This deliverable does two things.

1. It lays out an algorithm in the same family as Hierarchical Task Network planning (HTN-Planning) that uses Combinatory Categorial Grammar's (CCG) to represent the plan libraries. This heavily leverages our prior work on plan recognition. In fact, the new planner makes use of the exact same representation. This means that we can use the same plan libraries to both recognize and build plans based on the same knowledge.
2. It then sketches a simple method for learning the plan CCGs based on some significant simplifying assumptions (principally that there is only a single unknown action in the plan). This method, critically uses the fact that the same CCG representation is used for both recognition and for building plans.

Thus, this deliverable sketches both a new planning algorithm and how domain specific knowledge necessary for the use of the planner and the existing plan recognizer can be leaned from a single process.

Chapter 2

Content

2.1 Abstract

Prior work has formally identified the relationships between context free grammars and hierarchical (HTN) planning. This paper presents a new hierarchical planning algorithm that formalizes the plans in the plan library using a gramatical formalism taken from natural language processing research, specifically combinatorial categorial grammars. This representation and new algorithm has three significant advantages, first, the representation explicitly represents the order in which subgoal planning should be performed allowing for a more fine grained control of the planning search. Second, it replaces searches for subplans at multiple levels of abstraction with the building of a single grounded plan at the level of basic actions while still being informed by the information in the hierarchical models. Third and finally, the representation enables a bidirectional, divide and conquer algorithm for planning that can identify early on if a particular abstract plan will fail.

2.2 Introduction

The AI planning problem stated simply is, given a model of the world, and set of *actions* or *operators* that transform one state of the world model to another, find a sequence of *operator* applications that when starting at a given initial state, reach a goal state. Within the Artificial Intelligence planning research community, there are two major kinds of planners. First, there are *traditional* (sometimes called linear) planners. This set would include the popular FF plannerHoffmann & Nebel (2001) and its variants. Such planners can be thought of as doing a brute force search through the space of possible operator sequences (sometimes, as in the case of FF, informed by the structure of the operators themselves). Second there are *hierarchical* planners, where in the search through the space of operator sequences is directed by additional domain specific information usually contained in structures called *methods* that are intended to capture abstractions about "usual" methods for achieving the goal.

There is a well known relationship between traditional planners and regular grammars and hierarchical planners and context free grammars(CFGs).Erol, Hendler, & Nau (1994). The operators can be seen as terminal symbols, and the methods as productions within the grammar. In this view, the planner can be seen as searching through the space of possible strings of operators produced by such a grammar for one that achieves the goal from the initial state. However, researchers working on formal theories of grammar have developed new formal grammars that are more expressive than CFGs while still having low complexity of parsing. In this paper, we provide an alternative formulation of planning based on a new formal grammar model and argue that it has a number of properties that make it attractive.

Beyond the intellectual exercise of building such a reformulation, one might reasonably ask why should we do it? The simple answer is that traditional HTN planning has foundational issues that have yet to be resolved, that can be addressed by reformulating it. The formost of these is the status of methods. Within the literature on HTN planning while they have been provided with definitions their actual formal status has not been well grounded. It is clear that they are not actions in the same way that basic operators are, and they don't represent single states of the world. They have been described as *abstract actions*,

however it is far from clear what this would mean, and what the relationship is between abstract and non-abstract actions.

From their earliest introduction in planning systems Sacerdoti (1974) it is clear that methods are intended to speed up the process of planning. Further the very popular use of HTNs in actual deployed systems (specifically to control search) is more than reason enough for us to believe they are effective in this regard. However it is clear that having the wrong methods may not only slow down the planning process it may in fact prevent the system from finding a solution.

HTN methods introduce a new problem space in which the problem must first be solved, however the formal relationship to the original problem domain is frequently left completely unspecified. Further, it is not at all clear where methods come from and how they are to be learned. As a result, HTN methods are frequently considered "domain knowledge", as such researchers working in *domain independent planning* argue that it should have to be learned, but its not at all clear how to learn HTN methods, and this is an active area of learning research.

Thus, in the following paper, rather than using a Context Free Grammar, we are going to use the ideas behind lexicalized grammars, specifically Combinatory Categorial Grammar (CCG) Steedman (2000) as the foundation of a new way to represent plans and then develop a new algorithm for building plans based on this representation. The organization The rest of this paper is organized as follows. First we will provide a more complete background on CCGs and how they can be used to represent action grammars, then we will discuss how they must be extended and an algorithm for their use in planning. The paper will then briefly outline a simple method for learning plan CCGs based on plan recognition with simple plans.

2.3 Plans as Grammars

This work is motivated by *embodied systems* Brooks (1999) theories about intelligent agents. As such, it is critical that our formalism take seriously the constraints imposed by execution of actions in physically embodied systems. Consider a robot or other physically realized agent. Such agents must have a continuous control system that drives its end effectors. For such a system, what has been called an *action* in the AI planning literature is nothing more than a motor program: a timed series of voltages to motors that, in the absence of other obstacles, will cause the end effector to move through a particular trajectory. For the rest of this document, we will use the word *action* only in this sense. For this paper the term "action" will be use as a synonym for a motor program or a basic motion primitive that is available from the embodied agent.[1] A plan then is nothing more than a ordered sequence of such actions, and the objective of the planning process is to generate such a sequence that will result in a particular state.

With this understanding of actions, it is worth noting, that there are no conditions that must be true (*preconditions* in the planning literature) before actions or plans can be run. There are also no conditions that must be true after the actions execution (*effects* from the planing literature). An action or plan is executed and a state of the world eventuates. Perhaps this is a desired state, but but our attitude toward the state has no necessary relation to the definition of the action. In other situations other states result from executing the program, but the program is the same. With this understanding, we make a strong commitment that any knowledge of states of the world that eventuates from the execution of an action is NOT part of the definition of the action but rather causal knowledge about how the world works.

With this in mind, we believe that a significant problem with prior work in HTN planning is a fundamental confusion about what kind of domain specific knowledge should be stored. The use of abstraction Sacerdoti (1974) and decomposition Tate (1977) in planning that were eventually formalized in HTN planning was to accelerate the planning process. In this paper we renew this commitment and suggest that the crucial piece of domain dependent information that should be maintained and learned is how to generate plans.

In this work, we propose to represent knowledge about how to generate plans using a particular *lexicalized grammar* called Combinatory Categorial Grammars (CCG) Steedman (2000). Unlike more traditional grammars, like context free or regular grammars, where the domain specific constraints specific of the

[1] Note this position does not preclude the set of actions changing over time. Imagine learning to ice-skate or play an instrument. To do these things an agent must learn new motor primitives. That said the position taken in this document is that high level reasoners do not have the ability to build or define new actions. As we will see it can only build abstractions over the actions it already has. Learning new actions is the provenance of the agent's lower level continuous controller.

language are spread between the rules of the grammar and the lexicon, lexicalized grammars move all language-specific information into rich data-structures called *categories*. Such categories are associated, by a lexicon, with individual actions that an agent can execute and crucially the system can recognize as being run.

In CCGs we define the categories recursively as:

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

Complex categories : $\forall Z \in \mathcal{C}$, and non empty set $\{W, X, ...\} \subset \mathcal{C}$ then $Z\backslash\{W, X, ...\} \in \mathcal{C}$ and $Z/\{W, X, ...\} \in \mathcal{C}$.

The intuition behind these categories is that, like the actions they represent, categories are functions. They are functions that capture knowledge about how to build a plan to achieve a given single domain predicate that is associated with a category. Basic categories are simple functions that have no arguments. As such, associating them with an action in the lexicon suggests that executing the action can unconditionally result in the predicate being made true in the world.

Complex categories in contrast represent functions that take a set of *arguments* ($\{W, X, ...\}$) and produce a *result* (Z). The direction of the slash indicates where the function looks for its arguments and the order of the construction of the plan. We require the argument(s) to a complex category be constructed in the order they are presented in the category and executed in the ordered defined by the slashes and their specification in the category.

To provide some intuitions, the semantics of the "\" is roughly that of a sub-plan that must precedes the present category's action and the semantics of "/" is sub-plan that must succeed the current category's action to produce the result. Therefore, associating with an action, α the category $A\backslash\{B\}$ tells us that executing α can as part of constructing a plan for achieving A. However, to do so requires building and executing a plan to achieve B before executing α. Likewise, $A/\{B\}$ tells us that the action associated with this category can be executed as part of a plan to achieve A, but only if the execution of α is followed by a plan for achieving B.

Note this is at odds with traditional ideas about basic actions taken from the planning literature. The traditional definition of actions in terms of *preconditions* and *effects*, where preconditions define "when an action can be taken", and effects define "what results from the action" Russell & Norvig (2010). The formulation of knowledge about actions advocated here is divorced from these ideas. Categories as we have defined them here have no preconditions or effects. Rather they are statements about the effective uses of an action and the steps that must be taken to use the action to achieve that end. As such it is much more in line with thinking about actions within embodied systems.

As an example, consider a simple lexicon for two different plans for cellphone use, one for calling a friend and one for reporting a fire. Both of these plans call for getting the phone, opening it, placing a call, and talking. Therefore one lexicon for these plans is:

CCG: 1.

$$\begin{aligned} dial - cellphone :=& ((REPORT/\{T\})\backslash\{G\})\backslash\{O\} \mid \\ & ((CHAT/\{T\})\backslash\{G\})\backslash\{O\}. \\ talk - cellphone :=& T. \\ get - cellphone :=& G. \\ open - cellphone :=& O. \end{aligned}$$

Where $G, O, T, REPORT$, and $CHAT$ are basic categories, and the observation of the action *dial* has two complex categories assigned to it by the lexicon: one for reporting a fire and one for chatting to a friend. To help in or future discussion we will define two pieces of terminology. First, define the *root result* of a category as the left most, inmost, basic category. For example, $REPORT$ and $CHAT$ are the root results of the two categories assigned to the action $dial - cellphone$. Second, define an action α as an *anchor* for a plan to achieve C just in the case that the lexicon assigns to α at least one category, who's root result is C.[2]

[2]For the purposes of this discussion we have used a propositional representation. This is not actually necessary. We could have used a first order representation, however the action arguments would be largely a distraction. Where necessary we will mention their addition.

Keeping in line with the thesis that what should be represented is knowledge about how to construct plans, we add one more requirement on complex categories. The plans for each of its arguments must be built in the order they are provided by the category. Thus using our example lexicon. If we want to use *dial* as the anchor for a plan to $CHAT$, the category requires not only that we build subplans for getting the cellphone, opening it, and talking, but these plans must be built in the order of: 1) opening the cellphone, 2) getting the cellphone, and then 3) talking. Thus, unlike other hierarchical planners, the category specifically enumerates not only the subplans that have to be built but the order in which they should be built.

Using such a grammar we can imagine a simple algorithm to generate sequences of actions that meet the specifications of the grammar. For any category, G, that we want to achieve we can search the space of expansions of categories that have G as their root result. Such an algorithm is captured in the pseudo-code shown as Algorithm 1. The function buildPlan has two inputs: a desired goal category (g), and a lexicon (l), and a single output: an ordered list of actions (p) to achieve g.

Algorithm 1 Simple Planning Algorithm

```
1: procedure BUILDPLAN(g, l)
2:     (a, c) ← choose( rootResult( g, l ) )
3:     p ← [a]
4:     while ( isComplex( c ) ) do
5:         left ← isLeftward( c )
6:         c′ ← popFirstArg( c )
7:         p′ ← buildPlan( c′, l )
8:         if ( left ) then p ← p + p′
9:         elsep ← p′ + p
10:        end if
11:    end while
12:    return p
13: end procedure
```

This pseudo code makes use of the following functions:

- *rootResult(g, l)*: Returns the set of all lexical assignments that have category "g" as their root result.
- *isComplex*: Returns true when given a complex category and false otherwise.
- *isLeftward*: Returns true if given a complex category whose first argument is to the left and false otherwise.
- *popFirstArgument*: Destructively modifies its complex argument category by removing and returning its first argument category.
- $+$: Is the traditional append function for lists, and finally
- *choose*: Returns pair made of an action (a) and a category(c) from the filtered set of lexical items that are are given to it.

The simple algorithm given here will produce a plan that should achieve the input category and meet the requirements of the grammar. However, note that simply generating such a sequence of actions is insufficient. The choose function could return any of the possible lexical entries and all such possible parses would meet the requirements of the grammar. But this doesn't mean they will be effective plans to achieve the goal.

The grammatical categories defined here are not intended as a specification of causal knowledge, but rather as a specification of a way to build a plan that can result in the desired root category being achieved. We need to consider the state in which the plan will be executed.

As is well known in HTN planning, unless the world state is modeled and method use is modeled in a particular world state, having a plan that meets the requirements of the grammar does not guarantee that the desired effects will actually be achieved. The grammar provides a specification for how to go about building a plan, it does not guarantee that the plan will be effective. In order to determine if the

plan will be effective. We need to add knowledge about the effects of actions and to be able to project the effects of sequences of actions in order to predict the state of the world that will result from executing a series of actions. In the following section we add this knowledge to the action grammar.

2.4 Projecting Action Sequences

In addition to the gramatical categories, we will associate with each action a set of *projection rules*. Each such rule defines a condition and a set of effects that occur if the action is executed in a world state that meets the condition. For our system this rule set is assumed to be exclusive and exhaustive so that only a single rule is applicable in any given state of the world. This eliminates the need for multiple rule invocations. If no rule is found that is applicable then the action is assumed to have no effect on the state of the world. Note that this may not be accurate. Executing the action in such a domain may provide evidence for the learning of a new projection rule or the generalization of an existing rule.

Such a set of rules is most closely related to the idea of *secondary preconditions* Pednault (1989) from the planning literature. Notice that they define a causal model for the action, but they are not traditional preconditions. They don't define when an action can be used. Instead they define what the agent believes will happen if the action is executed in a particular state of the world. Thus, this approach separates the knowledge of how to build plans to achieve a given predicate and the knowledge about how to project an initial state of the world given a sequence of actions.

We note that, preconditions have also been suggested as a method for constraining the search space for a given plan. In addition to secondary preconditions, multiple kinds of preconditions have been defined. For example early work in planning Sussman, Winograd, & Charniak (1971) distinguishes preconditions that can and should be achieved by backchaining planning, those that cannot or should not be achieved in this way. Preconditions have also been used to indicate when a given action is likely to be effective as part of a plan and therefore should be used. However, this not what we advocate here. The rules advocated here simple define the causal model of the action, what is made true by its execution in differing contexts.

To summarize then, our proposal distinguishes and advocates three separate kinds of information about actions.

1. *action execution procedure*: The procedural linkage to lower level execution. This is the foundation of any action and is represented by a control program that is to be run on the system's execution platform.
2. *action projection rules*: The causal models for an action just described, as condition effect pairs. This information defines at an abstract level those facts about the world that will change as a result of an actions execution given the initial state of the world.
3. *action/plan lexicon*: The information about how plans are constructed that we have captured in CCG categories. Keep in mind this information does not define causal knowledge about the world (ie. how the world works) but rather captures knowledge about how successful plans have been built in the past.

Note that only the second two of these kinds of knowledge are used in the building of plans. Having the first kind of knowledge is critical to understanding how such a system learns but is not strictly necessary for building plans.

Much of the prior work on *domain independent* planning systems have taken such causal knowledge or *transition functions* as definitions of actions and searched through he state space defined by the specified transitions to find a plan using no other information. In contrast, HTN planning is largely seen as a form of *domain dependent* planning. The methods that drive HTN planning are seen as domain dependent knowledge because their effectiveness in speeding up search often depend crucially on the domain they are applied in.

However given the difficulty of applying domain independent planners to real world problems, and the effectiveness of HTN methods in actual deployed systems, we would argue that uninformed searching of the kind done in domain independent planning is misguided. For any real world embodied system this space produced is far to large for uninformed search. Even if the number of actions the agent can perform is small, the variability of the world will make the space of states that might reasonably result from the

action's execution very large. This means that further information about how to search for an effective plan will be necessary if planning is to proceed efficiently.

In the next section we will provide an algorithm for using a plan lexicon and a set of projection rules for a given set of actions that we have just talked about for building plans.

2.5 Building Plans

We will make three significant changes to the previous *buildPlan* algorithm. First the algorithm will itself now take a third argument (i) representing the initial state that the plan is to be built from.

Second after plan construction we test that the plan that is produced will actually result in the desired category being achieved. This is captured in line 12 of Algorithm 2. Here the function *project* which takes a plan and an initial state is assumed to return the state that will result from the execution of the plan staring in the initial state. In line 12 we see a test to see if the goal state is entailed by the state that results from executing the plan in the initial state.

A short digression: We could imagine adding to the complexity of the algorithm by making the projection process probabilistic. Rather than the exclusive and exhaustive set of deterministic rules that we have outlined above, the model for each of the actions could be a probability distribution over the set of possible states that could result from its execution in a given state. We could then reformulate project to produce a probability distribution over the set of possible resulting states that would result from executing the plan. We can then compute the likelihood of the final state satisfying the goal category by simply summing the probability mass associated with those possible states that did satisfy the goal. We might then want to search through the set of all plans to make sure that the one we executed maximized the probability of success. While such an extension is relatively straight forward, it is computationally much more costly, and to simplify this discussion we will assume a deterministic model of actions and method of determining if the plan satisfies the goal category.

The third and final change to the code actually is just a redefinition of the role of the *choose* function. In Algorithm 1 the role of the choose function was to select any of the action category pairs from the set passed to it. In the case of Algorithm 2 we will redefine *choose* as an oracle that will always choose the correct pair that will lead to a viable plan starting in the initial state. This makes Algorithm 2 non-deterministic and allows us to avoid explicitly coding the search though the space of possible expansions in the pseudocode. In our actual implementation of this algorithm we have built the choose function as

Algorithm 2 Planning Algorithm

```
1:  procedure BUILDPLAN(i, g, l)
2:      (a, c) ← choose( rootResult( g, l ) )
3:      p ← [a]
4:      while ( isComplex( c ) ) do
5:          left ← isLeftward( c )
6:          c′ ← popFirstArg( c )
7:          p′ ← buildPlan( i, c′, l )
8:          if ( left ) then p ← p + p′
9:          elsep ← p′ + p
10:         end if
11:     end while
12:     if ( project( p, i ) ⊢ g ) then return p
13:     else return []
14:     end if
15: end procedure
```

a search and multiple possible search strategies are possible in this space. This completes our high level discussion of the planning algorithm. However there are a number of properties that this algorithm has that are worth noting.

First, as with other state of the art planning algorithms, this algorithm produces a totally plan. That is, the plan output by the process will be a totally ordered set of actions. We have claimed that the CCG categories only define information about how to go about building the plan. However, because

CCGs have a nested structure and provide ordering information, in the form of the directionality of the arguments, they also enable the production of totally ordered plans.

Second, and more interestingly, at each iteration of this planning algorithm an action is added to the plan. This is in contrast to some HTN planners that must first build a complete abstract plan before further refining the plan using more methods. To see this, consider that each CCG category can be thought of as a tree spine Aho & Ullman (1992). The action the lexicon associates with the category is its leaf and the root result is the tree's root with nested category arguments being non-terminals at higher levels of the tree either to the left or the right of the spine. With this view, the process of planning can be intuitively thought of as a form of tree adjunction Joshi (1985) or composition. A root tree is chosen adding the first action to the plan (possibly somewhere in the middle of the plan) and later trees that are added to the plan both add actions to the plan and are spines that resolve the argument categories (non-terminals) of the original plan tree.

Third and finally, to the best of our knowledge, this is the only planning algorithm that is able to build plans "from the inside out", that is building the plan from the middle of its execution outward. After a category is chosen, the action associated with it is added to the plan. Future actions, that are designed to address the category's arguments, can be added either before or after that action. Effectively this means that depending on the structure of the category the middle of the plan can be built first in a sort of "divide and conquer" approach to plan construction.

This plan construction is directed by structure of the CCG categories. CCG categories can be built that are all rightward looking and therefore result in a forward search of the plan space. They can also be build with all leftward looking categories resulting in constructing the plan backwards from the goal state. Or as we have already discussed, categories can direct the building of the plan to alternate sides and effectively build the plan from the middle outward.

It is also worth noting that at any given moment the plan is executable. Categories are not part of the plan but rather direct its construction. Therefore at any time the current plan is a meaningful, possibly only partial, plan to achieve the goal. There are no "abstract actions" in the plan. This is very important for being able to test the plan for executability and success at achieving its result categories.

We have implemented a propositional version of this planning algorithm to prove its viability for simple problems. However for acceptance within the planning community, a propositional version is insufficient. It is necessary for the planer to support at least a limited form of first order representations of both states and actions. We are in the process of completing the extension of the implementation to include this more expressive representation. While this does not represent a great theoretical leap forward for the system, it does greatly complicated the book keeping necessary for the search among possible plans since not only multiple categories need to be considered by multiple instantiations of the categories and related actions must be considered both in the search and in the projection of actions to verify successof the resulting plans.

2.6 Recognizing Plans Using CCGs

With such a representation of actions and planning system in hand, the natural question to be asked is can such plan grammars be learned automatically rather than being built by hand. We believe that they can, and that this ability depends on the fact that the identical representation of actions, here used for planning, can be used for plan recognition.

Briefly, plan recognition is the identification of an agents high level plans and goal based on observations of their actions and a library of plans to be recognized. In fact, we have already documented the effectiveness of CCG lexicons for plan recognition in prior work Geib & Steedman (2007); Geib (2007, 2009); Geib & Goldman (2011). This work viewed plan recognition as probabilistic parsing of a plan CCG. We will refer the interested reader to the prior work for a detailed understanding of this process, however some intuitions about how this process is done will help in understanding our proposed learning process.

Given a CCG lexicon for a set of actions, and a sequence of observed actions, we can parse the action sequence based on the lexicon to produce a (possibly singleton) sequence of categories that capture the plans that the agent was executing. For example, consider the example lexicon shown in Figure 2.1. In this case, we have five observable actions a, b, c, d, e, and a lexicon that has categories for each of them.

Lexicon:	$a = A, b = B$ $c = (((G/E)/D)\backslash A)\backslash B$, $d = D, e = E$
Observed action time = 1:	a
Parse (initial):	$[A]$
Observed action time = 2:	b
Parse (initial):	$[A, B]$
Observed action time = 3:	c
Parse (initial):	$[A, B, (((G/E)/D)\backslash A)\backslash B]$
Parse (left apply B):	$[A, ((G/E)/D)\backslash A]$
Parse (left apply A):	$[(G/E)/D]$
Observed action time = 4:	d
Parse (initial):	$[(G/E)/D, D]$
Parse (right apply D):	$[G/E]$

Figure 2.1: Example of plan recognition based on parsing a set of observations using a CCG lexicon.

One category (the one assigned to c) describes a plan to achieve the goal category G. We can identify this because G is c's categories root result.

Assume as in Figure 2.1, that we observe the actions $[a, b, c, d]$ in that order. When we observe the a it can be assigned its lexical category of A, likewise b is assigned B, and c is assigned $c = (((G/E)/D)\backslash A)\backslash B$. However with this category we can combine c with b and then with a using nothing more than function application. Because c's category is a complex category and both A and B are leftward arguments to it, and they are in the correct order with respect to each other and c, they can be removed by leftward function application, resulting in $((G/E)/D$. When the next action that is observed is d and is assigned the basic category D, it too can be combined. This time the algorithm would use rightward function application producing the category G/E.

Thus as each action is observed, it is assigned a category and where possible these categories are combined into simpler categories to represent the component actions taking part in a single plan. In the case of Figure 2.1 to achieve goal G. The state of the inferred plan at the end of that example also suggests that the the plan is incomplete. It is still awaiting the execution of an e.

2.7 Learning CCG Plan Lexicons Through Bootstrapping

In order to leverage this kind of plan recognition to learning new categories for actions, we will view this as a case of high level learning by demonstration. As such, we will assume there is a teacher that presents the execution of a plan for a known goal that makes use of a single previously unknown action. This means that in our examples we will be learning one action at at time and that it is a wholly unknown action (ie. it has never had a category assigned to it before.)

Since there is only one action that is unknown, it means that all of the other actions have at least one categories assigned to them by a plan lexicon and using the plan recognition algorithm we can determine the most likely category for all of the other actions. Further we can also assume that we know the root result of the plan since a teacher is present and can label the training instance. We also know if any of the categories for the known actions have the goal of the demonstration as their root result.

Under these assumptions, our job is to assign to the new action a category that will allow us to recognize another instance of this plan if it is presented again. We break this process down into two cases.

In the first case, once plan recognition has been performed on the observed action sequence, the goal category for the demonstrated plan is already the root result of the category assigned to one of the known actions in the demonstration example. Effectively what this tells us is that the unknown action (possibly in conjunction with other actions in the plan) is playing the same part as a known category within the plan. Consider the case shown in Figure 2.2.

In this case, after plan recognition, the complex category in the explanation for the observed actions

Teaching goal category:	G
Lexicon:	$a = A, b = (((G/E)/D)/C)\backslash A,$ $d = D, e = E$
Observed actions:	$[a, b, x, d, e]$
Results of parsing:	$[((G/E)/D)/C), UNKNx, D, E]$
Inferred category:	$x = C$

Figure 2.2: Example of inferring a category for a previously unseen action when the the most likely parse of the observed actions results in a category with the goal root result as a category.

Teaching goal category:	G
Lexicon:	$a = A, b = B, d = De = E$
Observed actions:	$[a, b, x, d, e]$
Results of parsing:	$[A, B, UNKNx, D, E]$
Rightward arguments:	$[E, D]$
Leftward arguments:	$[A, B]$
Inferred category:	$x = (((\alpha/E)/D)\backslash A)\backslash B$

Figure 2.3: Example of inferring a category for a previously unseen action when the the most likely parse of the observed actions does not result in a category with the goal root result as a category.

does account for the actions a and b and the previously unseen action x is given a placeholder category ($UNKNx$). However, the D and E categories are being prevented from being combined with the complex category by the absence of a C category in the plan. If only the lexicon assigned to action x had the category C, the entire parse would have gone through and the plan been recognized. In the simplest such cases the only categories that have to be considered are the current next argument for the complex category since their removal is what is standing in the way of the other actions automatically becoming part of the plan.

Thus, all that is required in this case is for us to:

1. Recognize that one of the categories in the most likely parse of the plan has as its root result the goal category.
2. Recognize that some subset of the actions that follow the unknown action would be applicable to the recognized plan fragment.
3. Assign to the unknown action the category that will enable the rest of the plan to be completed.

In the second case, the goal category for the demonstrated plan is not the root result of one of the other actions that has been observed. This is in fact easier to handle that then previous case. All we need do is create a new category for the action that has two sequences of arguments. The first captures the ordered list of categories found to the unknown actions left, and the second the categories for the actions found to its right. The root result for the category is the category the teacher has already told us they are demonstrating. The elements of the two lists are then added to the category working from left to right (fist executed to last) in the case of the leftward arguments, and working from right to to left (last executed to first) in the case of the rightward arguments. Note if desired the the left and right arguments can be interleaved, but to make plan recognition easiest (and to recognize when earlier substeps of a plan will fail) it is generally best for the leftward arguments to be on the "outside" of the category (added to the category after the rightward arguments). Figure 2.3 shows an example.

If this process is successful and the new category is learned, not only can future instances of this plan be recognized the planner we have described isn now in a position to build plans that make use of the action in order to learn the projection rules for the action and complete the building of the action model and plan lexicon.

2.8 Conclusions

This paper has sketched how planning can be accomplished based on Combinatory Categorial Grammars (CCGs) and how learning of these same grammars can be driven by plan recognition based on this same type of grammar. We have already demonstrated the viability of creating new categories for previously unseen actions

Bibliography

Aho, A. V., and Ullman, J. D. 1992. *Foundations of Computer Science*. New York, NY: W.H. Freeman/-Computer Science Press.

Brooks, R. 1999. *Cambrian Intelligence: The Early History of the New AI*. Cambridge, MA: MIT Press Inc.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239 and UMIACS-TR-94-31, Computer Science Department, University of Maryland, College Park, MD.

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

Geib, C., and Steedman, M. 2007. On natural language processing and plan recognition. In *Proceedings of IJCAI 2007*.

Geib, C. W. 2007. Using lexicalized grammars and headedness for approximate plan recognition. In *AAAI Workshop on Plan Activity and Intent Recognition (PAIR-2007)*.

Geib, C. 2009. Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. In *Proceedings IJCAI*, 1702–1707.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)* 14:253–302.

Joshi, A. 1985. How much context-sensativity is necessary for characterizing structural descriptions - tree adjoining grammars. In *Natural Language Processing - Theoretical, Computational, and Psychological Perspective*, 206–250. Cambridge University Press.

Pednault, E. P. D. 1989. Adl: Exploring the middle ground between strips and the situation calculus. In Brachman, R. J.; Levesque, H. J.; and Reiter, R., eds., *KR*, 324–332. Morgan Kaufmann.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artif. Intell.* 5(2):115–135.

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

Sussman, G. J.; Winograd, T.; and Charniak, E. 1971. Micro-planner reference manual. Technical report, MIT Artificial Intelligence Laboratory.

Tate, A. 1977. Generating project networks. In Reddy, R., ed., *IJCAI*, 888–893. William Kaufmann.