June 1992

A Structural Interpretation of Combinatory Categorial Grammar

James Henderson
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-49.

A Structural Interpretation of Combinatory Categorial Grammar

Abstract

This paper gives an interpretation of Combinatory Categorial Grammar derivations in terms of the construction of traditional phrase structure trees. This structural level of representation not only shows how CCG is related to other grammatical investigations, but this paper also uses it to extend CCG in ways which are useful for analyzing and parsing natural language, including a better analysis of coordination.

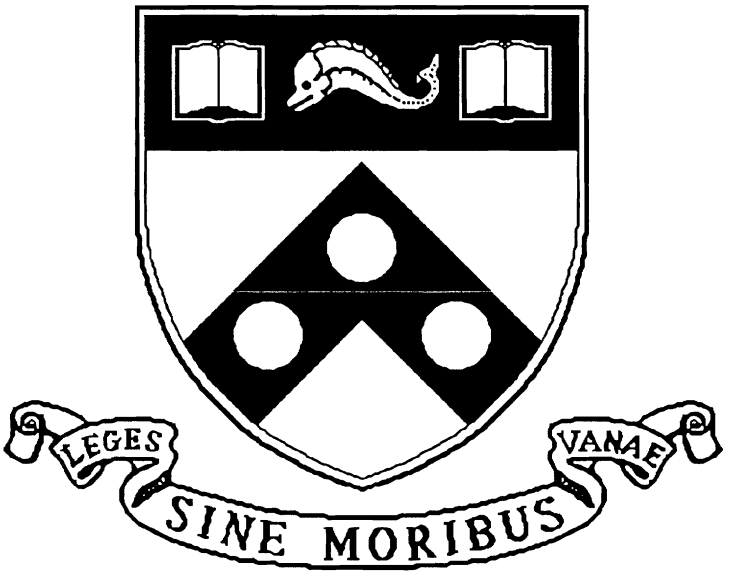
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-49.

A Structural Interpretation of Combinatory Categorial Grammar

MS-CIS-92-49
LINC LAB 227

James Henderson



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

June 1992

A Structural Interpretation of Combinatory Categorial Grammar

James Henderson

Abstract

This paper gives an interpretation of Combinatory Categorial Grammar derivations in terms of the construction of traditional phrase structure trees. This structural level of representation not only shows how CCG is related to other grammatical investigations, but this paper also uses it to extend CCG in ways which are useful for analyzing and parsing natural language, including a better analysis of coordination.

1 Introduction

Most investigations into natural language syntax have made extensive use of phrase structure trees which basically parallel predicate–argument structure. Combinatory Categorial Grammar (CCG, [Steedman, 1987]) does not use this traditional notion of constituency but instead uses the functional nature of phrases to determine constituency. This approach has led to theories of several phenomena, particularly coordination, which traditional approaches have had difficulty handling. However, in abandoning traditional constituency, CCG loses a level of representation which has proven very useful in investigating the nature of natural languages. The loss of this level of representation even complicates the analysis of coordination which has been so important to CCG's success. This paper argues that there is a simple structural interpretation of CCG categories that implies extensions which allow other linguistic work to be incorporated into a CCG theory, and which allow a better theory of some coordination phenomena.

The structural interpretation of CCG is based on the observation that CCG derivations can be thought of as constructing a Lambek calculus ([Lambek, 1961]) proof that the sentence is of type S. CCG categories are the minimal representation of Lambek calculus proofs, and the major CCG combination operations are rules which take two proofs and combine them to produce a third. If this system is redefined so that it doesn't use the minimal representation, but instead encodes all the significant information about a proof in the proof's category, these new categories have a natural interpretation in terms of traditional phrase structure trees. This is because a Lambek calculus proof provides all the semantically significant information about the derivation history of its CCG category. The structural interpretation of a CCG category is a canonical representation of this semantically significant information about the category's derivation history. This structural interpretation can be expressed as a partial phrase structure tree in a formalism very similar to Lexicalized Tree Adjoining Grammar (LTAG, [Schabes, 1990]). By adding this structural level of representation to CCG, the formalism can express and maintain information about the phrase structure of a derivation constituent, while still combining derivation constituents independently of that phrase structure constituency. Thus the notion of constituency used in CCG is still manifested in the derivation structures of this phrase structure based formalism.

The structural representation just discussed and CCG categories are the two extremes of a continuum which varies as more information about a Lambek calculus proof is included in the proof's category. The additional information is needed for such tasks as incrementally parsing posthead

modifiers, but such information can interfere with doing coordinations. Both these situations can be accommodated by adding a separate operation for abstracting away from details about a proof. This allows a derivation to keep information about a proof when it needs it, and forget information when that information becomes a complicating detail. In CCG the combination operations and the abstraction operations are conflated. By separating these operations we get an analysis of coordination which actually improves on that provided by CCG. In the last section this argument is supported by linguistic examples.

2 CCG and Lambek Calculus

The categories used in Combinatory Categorial Grammar are the same as the syntactic types used in Lambek calculus. These types represent the functional nature of constituents in derivations. For example, a transitive verb is given a category whose result is an S but which takes as arguments both an NP to its right and an NP to its left, written (S\NP)/NP. Lambek calculus is a proof system which defines what sequences of categories can be reduced to what single categories. The version of CCG which is assumed here respects these definitions in that it can only derive sequences of categories which are reducible to the category S[1]. This section presents Lambek Calculus and CCG, and shows why CCG derivations respect the definition of categories given in Lambek calculus.

2.1 Lambek Calculus

The syntactic types of Lambek calculus are either basic categories (such as NP or S) or of the form (X/Y), (X\Y), or (X·Y), where X and Y are syntactic types. In this notation X/Y is a type for a constituent which would be of type X if it combined with a constituent of type Y to its right. X\Y is the same except the Y is wanted on the left. X is called the result of the category and Y is the argument. The category X·Y is the concatenation of X and Y.

Lambek calculus is a Gentzen style proof system which uses one axiom and a set of inference rules to deduce subtype relationships. The sequents are of the form $\Delta \rightarrow X$, where Δ is a sequence of types and X is a single type. This sequent means that a sequence of things with the types and order specified in Δ are of the type X. The one axiom of the system is $X \rightarrow X$, which expresses the trivial equivalence of identical types. The inference rules are as given below. The sequent(s) above the line are the antecedents of the rule and the sequent below is the consequent. For example, the /L rule should be read "if Δ is of type Y and Γ,X,Λ is of type Z, then $\Gamma,X/Y,\Delta,\Lambda$ is of type Z". The Cut rule is not actually needed, since any theorem which can be proved with the Cut rule can also be proved without it. This Cut elimination result means that if Δ is of type X, then any reductions which X can participate in, Δ can participate in in the same way. The fact that this is true is why we can interpret this calculus as proving syntactic subtype relationships.

$$\textbf{/L:}\ \frac{\Delta \rightarrow Y \quad \Gamma,X,\Lambda \rightarrow Z}{\Gamma,X/Y,\Delta,\Lambda \rightarrow Z} \qquad \textbf{\textbackslash L:}\ \frac{\Delta \rightarrow Y \quad \Gamma,X,\Lambda \rightarrow Z}{\Gamma,\Delta,X\backslash Y,\Lambda \rightarrow Z} \qquad \textbf{/R:}\ \frac{\Delta,Y \rightarrow X}{\Delta \rightarrow X/Y} \qquad \textbf{\textbackslash R:}\ \frac{Y,\Delta \rightarrow X}{\Delta \rightarrow X\backslash Y}$$

$$\textbf{·L:}\ \frac{\Delta,X,Y,\Lambda \rightarrow Z}{\Delta,X\cdot Y,\Lambda \rightarrow Z} \qquad \textbf{·R:}\ \frac{\Delta \rightarrow X \quad \Lambda \rightarrow Y}{\Delta,\Lambda \rightarrow X\cdot Y} \qquad \textbf{Cut:}\ \frac{\Delta \rightarrow X \quad \Gamma,X,\Lambda \rightarrow Y}{\Gamma,\Delta,\Lambda \rightarrow Y}$$

[1]Most of the recent linguistic work done in CCG uses operations which actually violate the Lambek calculus definition of categories. However the core operations, function application and order preserving function composition, do not. These are the only operations allowed in the version of CCG being assumed here.

2.2 Combinatory Categorial Grammar

In CCG, grammars associate words with categories. These categories are the types defined in Lambek calculus. A CCG derivation maps each word in the sentence to one of its categories, and then combines these categories until the category S is produced. Several rules to combine categories have been proposed in CCG, but the primary ones are function application and order preserving function composition. The four cases of these rules are "X/Y Y → X", "Y X\Y → X", "X/Y Y/Z → X/Z", and "Y\Z X\Y → X\Z". With these combination rules coordination can largely be handled with the simple rule "X and X → X". Intuitively, two derivation structure constituents can coordinate if they can perform the same syntactic function in a sentence. By analyzing coordination in terms of these derivation structure constituents rather than in terms of traditional semantically determined constituents, CCG is able to treat many cases of "nonconstituent coordination" as simple constituent coordination. For example in the CCG derivation given below, "Barbie pushed" and "Ken rode" are each constituents even though they include a subject and only part of a verb phrase. Each derivation step in the example is shown as a line with the categories which fit the left side of the rule above the line and the result of the rule application below the line.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Barbie}}{S/(S\backslash NP)}\quad\dfrac{\text{pushed}}{(S\backslash NP)/NP}}{S/NP}\text{comp}\quad\text{and}\quad\dfrac{\dfrac{\dfrac{\text{Ken}}{S/(S\backslash NP)}\quad\dfrac{\text{rode}}{(S\backslash NP)/NP}}{S/NP}\text{comp}\quad\dfrac{\dfrac{\text{the}}{NP/N}\quad\dfrac{\text{tonka}}{N}}{NP}\text{app}}{S/NP}\text{coord}}{S}\text{app}$$

All the CCG rules being used here are theorems of Lambek calculus. Thus, for example, there is a Lambek calculus proof for the sequent X/Y,Y→X. Using this proof, a proof for the sequent Δ→X/Y and a proof for the sequent Γ→Y can be combined to produce a proof for the sequent Δ,Γ→X. This combination can be done using two applications of the Cut rule as follows:

$$\dfrac{\Delta \rightarrow X/Y \quad \dfrac{\Gamma \rightarrow Y \quad X/Y, Y \rightarrow X}{X/Y, \Gamma \rightarrow X}\text{Cut}}{\Delta, \Gamma \rightarrow X}\text{Cut}$$

The same technique can be used for any CCG combination rule "X Y → Z" to combine a proof of Δ→X and a proof of Γ→Y to produce a proof of Δ,Γ→Z. Given this fact, CCG categories can be interpreted as incomplete representations of Lambek calculus proofs, and CCG combination rules can be interpreted as combining two proofs to produce a third. A category X can be used to represent any proof which has X on the right side of its theorem. A combination rule "X Y → Z" ensures that any proof represented by X can be combined with any proof represented by Y to produce a proof represented by Z. Furthermore, if X represents a proof with theorem Δ→X and Y represents a proof with theorem Γ→Y, then the proof which Z represents has theorem Δ,Γ→Z. Thus if we assume initial categories in a CCG derivation represent the trivial proofs of the form X→X, then whenever a derivation combines a list of categories Δ, the resulting category represents a proof with the left side of the theorem being Δ. Therefore the S category which results from a derivation represents a proof of Δ→S, where Δ is the initial sequence of categories for the sentence. A simple example of a CCG derivation is given below with the whole theorem of the proofs shown, rather than just the right hand side of the theorem.

$$\dfrac{\dfrac{\dfrac{\text{Barbie}}{S/(S\backslash NP) \rightarrow S/(S\backslash NP)}\quad\dfrac{\text{rode}}{(S\backslash NP)/NP \rightarrow (S\backslash NP)/NP}}{S/(S\backslash NP), (S\backslash NP)/NP \rightarrow S/NP}\text{comp}\quad\dfrac{\dfrac{\text{the}}{NP/N \rightarrow NP/N}\quad\dfrac{\text{tonka}}{N \rightarrow N}}{NP/N, N \rightarrow NP}\text{app}}{S/(S\backslash NP), (S\backslash NP)/NP, NP/N, N \rightarrow S}\text{app}$$

3 Full Encodings of Proofs

The previous discussion of CCG showed that a CCG derivation can be characterized as constructing a proof that the initial sequence of categories for the sentence is of type S. However, the only information which a derivation maintains about the proof being constructed is the right hand side of the theorem. This raises the question of whether that one category is all a formalism needs to represent about a sequence of words in order to characterize natural language syntax. Clearly there are many different proofs which have the same right hand side of their theorem. Is it reasonable to put all these proofs in the same equivalence class? To address this issue this section presents a formalism where only semantically equivalent proofs of a given theorem are put in the same equivalence class. In other words, all information about a proof of a theorem which might be pertinent to the semantic interpretation of the sentence is represented in the categories of the formalism. This is the other extreme of the approach taken in CCG, where only the information which is necessary to ensure the existence of a proof for the desired theorem is represented in categories. This alternative approach is particularly interesting because such a representation of Lambek calculus proofs has a direct interpretation in terms of traditional phrase structure trees[2]. This connection with phrase structure based linguistic investigations shows how insights from each representation can be applied to the other. It also inspires the extensions to CCG to be discussed in the following section.

3.1 The Formalism

The semantic interpretation of a constituent is dependent on the categories assigned to the words of the constituent, and the unifications done between the semantic interpretations of the basic categories in those categories. If the constituent "rode tonkas" is given the sequence of categories "(S\NP)/NP, NP" and the derivation proves the theorem "(S\NP)/NP, NP $\rightarrow$ S\NP", then the semantic interpretations of the second two NP's have been unified in the semantics of this constituent. For other theorems it is not so easy to tell what semantic interpretations have been unified. For some semantically ambiguous constituents it is actually impossible to tell from the syntactic categories alone. For this reason this formalism's representation of proofs includes information about what basic categories in a theorem have their semantic interpretations unified. This is notated using integer subscripts. To determine these subscripts all that needs be done is redefine the basic categories of Lambek calculus to include integer subscripts. This does not change the calculus in any way except to record information about the proof in the theorem. Now the axioms are all of the form $X_i \rightarrow X_i$, or some more complex category with the same subscripts on both sides. This ensures that basic categories which have their semantic interpretations unified will have identical subscripts. In addition, all subscripts must be distinct unless an axiom requires that they be identical. The new formalism uses theorems with subscripts to represent proofs, and this representation of proofs plays the role of categories.

As an example of how subscripts work in this Lambek calculus notation consider the proof of type raising given below. Note that the two S's on the right hand side of the theorem have identical

[2][König, 1989] gives a proof that a single syntactic tree can be constructed for any set of semantically equivalent Lambek Calculus proofs. König uses her syntactic tree representation to define a normal form for Lambek Calculus proofs, thus allowing more efficient parsing by avoiding constructing semantically equivalent proofs. She does not discuss how this structural representation relates to CCG or coordination. Any such normal form approach to eliminating spurious ambiguity will have difficulty handling coordination because the "spurious" derivations are necessary in order to construct some constituents which can be coordinated. Similarly, [Hepple, 1991] uses normal form derivations in a categorial grammar style formalism to avoid the spurious ambiguity problem, and thus he would have the same problem if he were to try to parse coordinations.

subscripts, and the NP's on the left and right side have identical subscripts.

$$\dfrac{\dfrac{NP_1 \rightarrow NP_1 \quad S_2 \rightarrow S_2}{NP_1,\ S_2\backslash NP_1 \rightarrow S_2}\backslash L}{NP_1 \rightarrow S_2/(S_2\backslash NP_1)}/R$$

An example of a derivation in this new notation is given below. As the derivation proceeds semantic interpretations are unified, and thus subscripts are set equal. This requires substituting one subscript for another in the result of each combination. When the derivation is done the resulting theorem shows the correct pattern of semantic interpretation coreference.

$$\dfrac{\dfrac{\begin{array}{cc}\text{Barbie} & \text{rode}\\ NP_1 \rightarrow S_2/(S_2\backslash NP_1) & (S_3\backslash NP_4)/NP_5 \rightarrow (S_3\backslash NP_4)/NP_5\end{array}}{NP_1,\ (S_2\backslash NP_1)/NP_5 \rightarrow S_2/NP_5}\text{comp} \quad \dfrac{\begin{array}{cc}\text{the} & \text{tonka}\\ NP_6/N_7 \rightarrow NP_6/N_7 & N_8 \rightarrow N_8\end{array}}{NP_6/N_7,\ N_7 \rightarrow NP_6}\text{app}}{NP_1,\ (S_2\backslash NP_1)/NP_5,\ NP_5/N_7,\ N_7 \rightarrow S_2}\text{app}$$

For the moment the combination operations for this formalism are the same as those for CCG, only translated so as to use to the new representation of categories. Thus this formalism is just a notational variant of CCG. The new notation is important for two reasons. As will be discussed in the following subsection, this notation provides a structural interpretation of CCG, which allows work in traditional phrase structure to be related to work in CCG. This notation is also important because it provides the additional information necessary to extend the operations of CCG in desirable ways. These extensions will be discussed in the next section, along with motivating examples for these extensions.

3.2 The Structural Interpretation

To show the structural level of representation which is implicit in the formalism just defined, this subsection presents another formalism which is essentially equivalent to the previous one. The only difference is that types are restricted to not be greater than second order. As is argued below, the additional expressiveness of these formalisms makes greater than second order categories unnecessary for natural language. This new formalism is very similar to both Lexicalized Tree Adjoining Grammar ([Schabes, 1990]) and Structure Unification Grammar ([Henderson, 1990]). Instead of Lambek calculus proof theorems, this formalism represents categories as tree fragments. As in the previous formalism, the operations which combine tree fragments are equivalent to operations which combine Lambek calculus proofs. And just as the previous formalism had to produce a proof that the initial sequence was of type S, a derivation is successful if it produces a complete tree with root S.

The tree fragments of this formalism can be defined by giving a translation from the extended theorem notation used above. Examples of such translations are shown in figure 1, where the theorems in the first line of the derivation have been translated into the trees in the first line of the combination. The subscripts are shown in the trees to illustrate the correspondences with the theorems. Each nonterminal node in a tree fragment corresponds to a distinct subscript in the equivalent theorem. The label of the node is the same as the basic categories which have the subscript. There are also terminals for each word in the constituent. Parent–child relationships (here called immediate dominance relationships) are determined by the categories on the left side of the theorem. The parent of a terminal is the innermost result of its associated type. For any node which corresponds to the subscript of an argument of a type or to the subscript of a result of an argument of a type, its parent is the innermost result of the type. These links manifest argument–result relationships. The right hand side of the theorem shows what other things are needed before the derivation can be complete. A node which corresponds to an argument or to the result of an

argument on the right hand side is specified in the tree as a substitution node. This means that a substitution must take place at this node before the derivation is completed. First order arguments on the right hand side are represented with dominance relationships. Dominance is the transitive closure of immediate dominance, and before the derivation is over all dominance relationships must be replaced by chains of immediate dominance relationships. For every first order argument on the right hand side there is a dominance relationship from the node which represents the result of the argument down to each node which represents an argument of the argument. Finally, these trees are completely ordered, so no links can cross and no node can change to a different side of another node. The ordering of nodes in the tree is determined by the directionality of slashes and ordering of arguments on the left hand side of the theorem. The nodes which represent arguments of rightward slashes are on the right side of the terminal and those which represent arguments of leftward slashes are on the left side. The first arguments correspond to the closest nodes to the terminal and the last arguments correspond to the farthest away. The directionality of slashes in first order arguments is represented with a notation on dominance relationships which specifies whether they must be instantiated from the left or from the right. For leftward slashes the dominance relationship must be instantiated from the right, and for rightward slashes the dominance relationship must be instantiated from the left[3].

$$\frac{\begin{array}{cccc}\text{Barbie} & \text{said} & \text{Ken} & \text{squeaks}\\ \text{NP}_2,\ (\text{S}_1\backslash\text{NP}_2)/\text{S}_3,\ \text{NP}_4 \rightarrow \text{S}_1/(\text{S}_3\backslash\text{NP}_4) & & & \text{S}_5\backslash\text{NP}_6 \rightarrow \text{S}_5\backslash\text{NP}_6\end{array}}{\text{NP}_2,\ (\text{S}_1\backslash\text{NP}_2)/\text{S}_3,\ \text{NP}_6,\ \text{S}_3\backslash\text{NP}_6 \rightarrow \text{S}_1}\ \text{app}$$

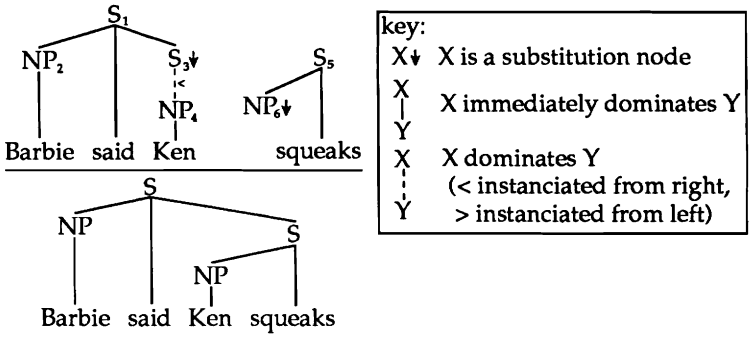


Figure 1:

The combination of two tree fragments always involves the substitution of the root of one of the trees for a node in the other. These substitutions can only occur at substitution nodes, but any tree fragment can be substituted as long as the root has the same label as the substitution node. If the substitution node has any dominance relationships below it, then the tree being substituted must have its own substitution nodes at which the dominated nodes can be substituted. This is the case in the example combination shown in figure 1. Also, such a secondary substitution node must be on the left side of the tree if the dominance link must be instantiated from the right, and on the right side of the tree if the dominance link must be instantiated from the left. Such a combination instantiates the dominance relationships involved, so they can be removed from the resulting tree. This instantiation of dominance relationships is analogous to adjunction in Lexicalized Tree Adjoining Grammar, and a dominance relationship is analogous to an obligatory adjunction constraint. Substitutions which do not involve dominance relationships are the same as

[3]Having this notation on dominance relationships is not natural, but it is necessary to represent the directionality of their corresponding slashes. There are reasons to believe that having to specify the directionality of these slashes is actually a handicap, and thus should not be represented.

substitutions in LTAG. The tree which results from a combination must have all the nodes in the same order as they were in the two original trees, and the order of the terminals must be the same as their order in the sentence.

This formalism constitutes a structural interpretation of CCG because its derivations are all equivalent to CCG derivations. As an example of the equivalence, the tree combination shown in figure 1 is equivalent to the CCG combination "S/(S\NP) S\NP → S". A tree fragment is equivalent to the theorem which it is a translation of, and a theorem is equivalent to the CCG category on its right hand side. From this fact we can see that the root of the tree is the innermost result of its equivalent CCG category, the substitution nodes without dominance relationships are the zeroth order arguments to the CCG category, and the substitution nodes with dominance relationships are the results of the first order arguments, with the argument's arguments being the dominated nodes. Whether a tree combination is equivalent to application or composition depends on whether the tree being substituted has substitution nodes which are not used in the combination. If the substituted tree has no substitution nodes after the combination, then the combination is equivalent to application. If a node in the substituted tree is still a substitution node after the combination, then the combination is equivalent to composition. If the tree combination does not instantiate dominance relationships, then the argument which is filled in the equivalent CCG combination is zeroth order. If the tree combination does instantiate dominance relationships, then the primary argument is first order. Thus the combination in figure 1 is equivalent to function application for a first order argument, as was indicated above.

The only CCG derivations which can't be done in this formalism are those which involve types which are greater than second order, as was mentioned above. New types of structural relationships could be defined which express higher order types, but that does not seem necessary for natural language. The only situations where higher order types are used in CCG is for modifiers of modifiers of first order types. For example in "Barbie very quickly became disillusioned", "very" has the category ((S\NP)/(S\NP))/((S\NP)/(S\NP)), which is third order. If the ability this formalism has to express the internal structure of constituents is used in the categories assigned in the grammar, then basic categories like VP, AP, and PP can be introduced, and modifiers of modifiers can be given categories like AP/AP or PP/PP. In this example if we introduce VP and AP categories we can give "quickly" the category "$AP_1, (VP_2\backslash AP_1)/VP_3 \rightarrow VP_2/VP_3$" and give "very" the category "$AP_4/AP_5 \rightarrow AP_4/AP_5$". One of the new rules to be introduced in the next subsection could then be used to combine these categories to produce "AP_1/AP_5, AP_5, $(VP_2\backslash AP_1)/VP_3 \rightarrow VP_2/VP_3$". This is an example of how the insights of phrase structure based linguistic investigations can be applied to a CCG style formalism through the above structural interpretation.

4 Extensions to CCG

So far the only justification for adding more information about Lambek calculus proofs to the categories of CCG is that it provides a phrase structure level of representation for CCG. This additional information has not been used in any combination operations. Also, the question of how coordination should be done with these expanded categories has not been addressed. This section first discusses combination rules which make use of the additional information in categories, then shows how the addition of abstraction operations allow the coordination rule "X *and* X → X" to be maintained.

In addition to providing a structural interpretation for CCG categories, viewing CCG derivations as constructing a Lambek calculus proof also has the advantage that the proof which results from a derivation provides all the semantically significant information about that derivation. Thus,

because the expanded categories of the new formalism represent all the semantically significant information about a proof, the expanded categories also represent all the semantically significant information about the derivation[4]. This means that not only can a combination rule extend previous derivations to derive another category, it can also modify previous derivations. Such a technique is used in [Pareschi and Steedman, 1987] to parse CCG efficiently in spite of the proliferation of semantically equivalent derivations in CCG. Because there are often many different ways to derive a given sentence in CCG, if a parser has to explore all possible ways it will not be efficient. An example of this difficulty is the problem of parsing posthead modifiers incrementally. For example in the sentence "Barbie likes jewelry with diamonds", after combining "jewelry" with "Barbie likes" the resulting category is S. Thus when "with" is reached there is no NP for it to combine with. To parse this sentence "jewelry" has to be combined with "with" before it is combined with "Barbie likes". With the expanded categories described above the NP for "jewelry" is still represented on the left side of the theorem which results from combining "Barbie likes" with "jewelry". Thus a single operation can calculate the affect of undoing the combination of "jewelry" with "Barbie likes", combining "jewelry" with "with", and then recombining the result with "Barbie likes". A rule for doing this operation is as follows:

$$\frac{\Delta, X_i \rightarrow Y \quad \Gamma \rightarrow (X_j \backslash X_k)/Z}{\Delta, X_k, \Gamma' \rightarrow Y/Z}\text{comp adj}$$

where Γ' is Γ with i substituted for j. The Z in this rule is there because "with" has another argument which needs to be passed on, as is done in composition. The real work in this rule is done by changing subscripts. First, the subscript of the X_i in the left antecedent needs to be changed to k to represent the fact that it now fills the argument slot for $X_j \backslash X_k$. Second, the X_j in Γ (which is the argument for the result of $X_j \backslash X_k$) needs to have its subscript changed to i to represent the fact that it is now filling the X_i argument role in Δ (previously filled by the X_i in the left antecedent). Hence X_i is changed to X_k and Γ is changed to Γ'. Another rule can be defined without the Z to allow for simple posthead modifiers.

The structural correlate of these new operations is analogous to optional adjunctions in LTAG. If a tree has a substitution node which has the same label as the root of the tree, then this tree can be adjoined at a node with the same label in another tree. The subtree under the node where the adjunction takes place is removed and substituted for the substitution node in the adjoined tree, and the result is substituted at the node where the adjunction takes place. The former substitution is the same as changing the i subscript to k, and the later substitution is the same as changing the j subscript to i. The biggest difference between this operation and adjunction in LTAG is that it can adjoin any tree with the requisite substitution node, whereas in LTAG the adjoined tree must be specified in the grammar as an auxiliary tree.

One problem with the way the above rule for attaching posthead modifiers was formulated in the theorem notation is that it can only apply to categories on the right edge of the left side of the theorem. This is because in order to enforce ordering constraints there can't be any types to the right of the X_i. This means modifiers can only attach at the lowest node in the tree, unlike the way it was specified for the structural version. For example, a prepositional phrase can't modify a transitive verb with this rule because the object of the verb will get in the way. Of course the presence of an object makes no difference for the attachment of a prepositional phrase, so we should be able to ignore the object and do the attachment. The ability to ignore insignificant

[4]Note that this is information about the derivation, not simply information about the semantics. CCG has a very elegant representation of semantic information, but semantic information would not be sufficient for our purposes. The extensions being discussed need to be sensitive to syntactic level constraints such as word order, which are not represented in the semantics. Including word order information in the semantic representation would be very strange, since word order is not pertinent to semantic interpretation.

details is also crucial to CCG's analysis of coordination. CCG has the ability to abstract away from details about a constituent and make coordination dependent only on the constituent's syntactic type. The new formalism as it has been described so far never abstracts. All information which is known at one point in a derivation is recorded in all subsequent points in the derivation using the left sides of theorems. In CCG, abstraction is always done as soon as possible, since CCG only records the information which is necessary to ensure a valid proof that the sentence is of type S. A compromise can be found between these two extremes by providing an abstraction operation which is independent from the combination operations. Thus a derivation can combine constituents without losing any potentially important information, and can still abstract when the information obscures the necessary generalization.

The simplest abstraction operation for this formalism is as follows:

$$\frac{\Delta, X/Y_i, Y_i, \Gamma \rightarrow Z}{\Delta, X, \Gamma \rightarrow Z}\text{app abst}$$

This operation abstracts away from the existence of the Y, and in doing so it does not jeopardize the validity of the final proof. Given any proof with the representation "Θ, Δ, X, Γ, $\Lambda \rightarrow$ W", a proof with the representation "Θ, Δ, X/Y_i, Y_i, Γ, $\Lambda \rightarrow$ W" can be constructed with the following step:

$$\frac{X/Y_i, Y_i \rightarrow X \quad \Theta, \Delta, X, \Gamma, \Lambda \rightarrow W}{\Theta, \Delta, X/Y_i, Y_i, \Gamma, \Lambda \rightarrow W}\text{Cut}$$

Thus given the final proof of a derivation, all abstractions can be undone in reverse order to produce a proof that the initial sequence of categories is of type S. Other abstraction rules can be defined which make use of other theorems of Lambek calculus, such as composition. The structural correlate of these operations removes a node from the tree and (in the case of composition abstraction) passes its children up to the node's parent. To be removed a node must be immediately dominated and not be a substitution node.

With an abstraction operation we can make use of the rule "X *and* X $\rightarrow$ X" to handle coordination. The X's in this rule are now the expanded categories of the new formalism, with the note that the actual subscripts aren't important, only the pattern of subscript equality. An example of how the abstraction rules allow this characterization of coordination to be maintained is shown below.

$$\dfrac{\begin{array}{ccc} \text{tonkas} & \text{and} & \dfrac{\dfrac{\text{red} \qquad \text{cars}}{NP_2/NP_3 \rightarrow NP_2/NP_3 \quad NP_4 \rightarrow NP_4}\text{app}}{NP_2/NP_3, NP_3 \rightarrow NP_2}\text{app abst} \\ NP_1 \rightarrow NP_1 & & \dfrac{}{NP_2 \rightarrow NP_2} \end{array}}{NP_1 \rightarrow NP_1}\text{coord}$$

An important question to ask when proposing an extension to CCG is how it affects the analysis of coordination, which has been so important to CCG's success. Two examples are given here which are pertinent to this question. The first, shown in figure 2, is an ungrammatical example of coordination which CCG syntactic categories can not distinguish from grammatical cases. The attempted coordination is between "Barbie" and the phrase "Barbie said Ken". In the CCG derivation, "Barbie" is given a type raised NP as its category, and the constituent "Barbie said Ken" is combined to produce the category S/(S\NP). These categories are identical, so the CCG coordination rule of "X *and* X $\rightarrow$ X" can apply to produce the category S/(S\NP). There is no way this CCG analysis of coordination can distinguish between this case and the grammatical example "Barbie said Ken, and Bill said Joe, squeaks". The crucial information which is needed

to distinguish these two cases is the coreference of the semantic interpretations[5]. In the category for "Barbie said Ken" the two S's are for two different clauses, while in the category for "Barbie" the two S's are for the same clause. In the analysis using expanded categories this difference is expressed, and thus the coordination rule can not apply. Even applying abstraction operations cannot reduce the right category to being the same form as the left category.

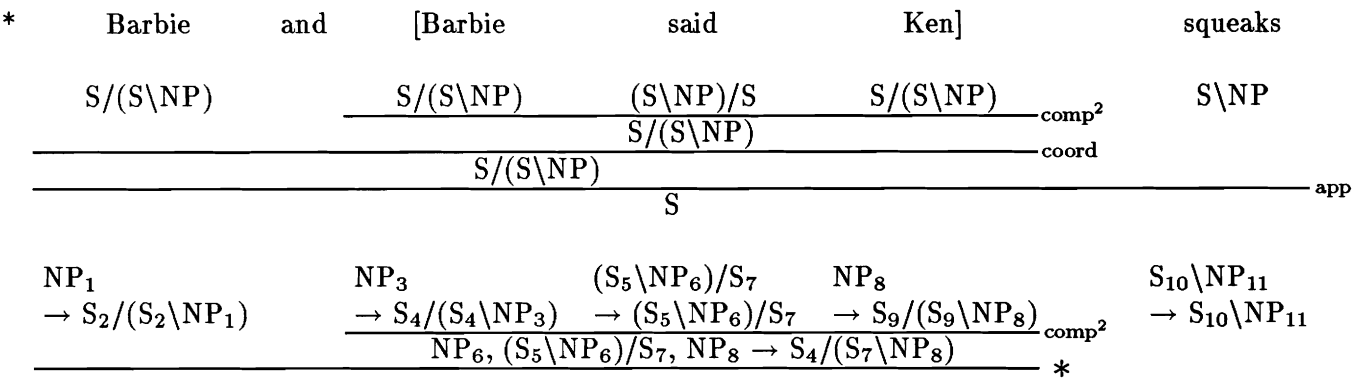


Figure 2:

The second example is a case where it is useful for a parser to maintain information about the derivation history of a pair of categories after they are coordinated. This example is shown in figure 3, with the coordination derivations given above the remainder of the derivations, for formatting reasons. The pertinent reading of this sentence is that both the belts and the shoes are in Barbie's wardrobe. This is another case of posthead modification, but in this case the need to coordinate forces the objects and the verbs to combine before the modifier can be combined with the objects. Otherwise only the shoes would be interpreted as being in the wardrobe. This problem can be solved by using a category for the NP objects which is type raised with respect to the NP\NP modifier, namely NP/(NP\NP). Then these categories can be composed with the verbs, the results can be coordinated, and the result of the coordination can apply to the posthead modifier, as shown in the example. Such an analysis creates nightmares for an incremental parser, which must type raise "the belts" long before it sees "in"[6]. A more natural analysis is possible with the expanded categories proposed above. Because the operations for the expanded categories allow combinations to be done independently from abstractions, the verbs can be combined with the objects without losing the information about the existence of the object NP's. The resulting categories can then be coordinated to produce a single category with one NP which represents both the objects. This category can then be combined with "in her wardrobe" using one of the

[5]This distinction is represented in the semantics of CCG categories, and Steedman (personal communication) has argued that this semantic information can be used to block this undesired coordination. This position requires the coordination schema to apply at a level of representation which includes syntactic categories plus semantic coreference of basic categories. The CCG categories with indexes used here are exactly such a level of representation (although Steedman would rather CCG's syntactic/semantic distinction be maintained). If abstraction operations are always applied as soon as possible, the extended version of CCG presented here is equivalent to a version of CCG which adds semantic coreference information to the categories. Thus the only empirical difference between Steedman's position and the one advocated here is whether abstraction should always be done as soon as possible. As was argued above and will be argued in the next example, efficient incremental parsing requires that abstraction be delayed in some cases. Since Steedman is primarily concerned with competence phenomena, this is not an issue for him.

[6]The affects of CCG's eager abstraction strategy can always be compensated for using type raising, as is done in this example. This is why a competence theory does not have to be concerned with the issue of when to do abstraction. However a theory which is concerned with performance constraints such as incrementality must address this issue, and, as is argued here, must permit abstraction to be delayed.

operations introduced above for attaching posthead modifiers.

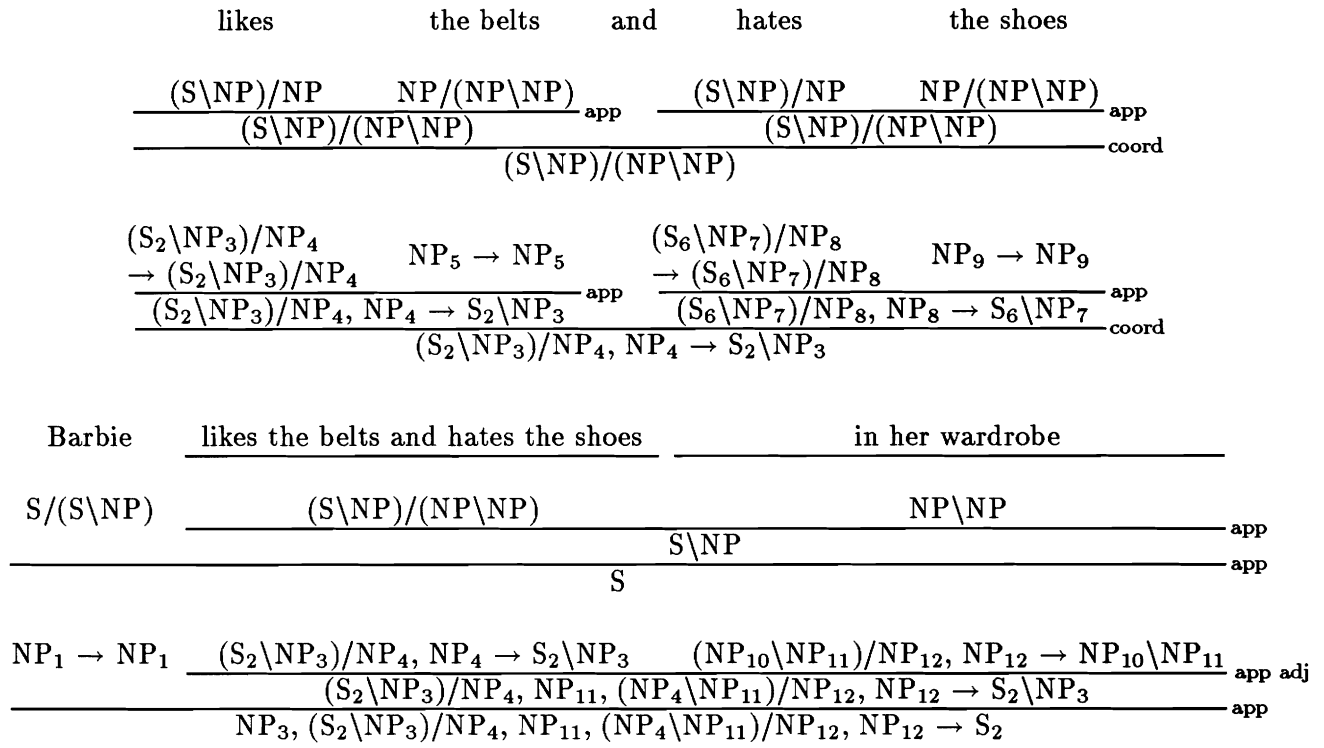


Figure 3:

5 Conclusion

This paper has given an interpretation of CCG derivations in terms of the construction of traditional phrase structure trees, and shown how this structural level of representation can be used to extend CCG in ways which are useful for analyzing natural language. CCG categories can be thought of as a representation of Lambek calculus proofs, and by incorporating more information about these proofs in categories it is possible to represent the semantically significant information about the derivation history of a category in the category itself. This added information constitutes a partial specification of a phrase structure tree. This structural information can be used to parse efficiently despite the proliferation of semantically equivalent CCG derivations. With the addition of an abstraction operation which is separate from the combination operations, this expanded formalism even improves on CCG's analysis of coordination. It is hoped that this connection between Categorial Grammar representations and phrase structure representations can lead to other fruitful interactions between these two areas of investigation in the future.

References

[Henderson, 1990] James Henderson. Structure unification grammar: A unifying framework for investigating natural language. Technical Report MS-CIS-90-94, University of Pennsylvania, Philadelphia, PA, 1990.

[Hepple, 1991] Mark Hepple. Efficient incremental processing with categorial grammar. In *Proceedings of the 29th Annual Meeting of the ACL*, pages 79–86, Berkeley, CA, 1991.

[König, 1989] Esther König. Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the ACL*, pages 272–279, Vancouver, B.C., Canada, 1989.

[Lambek, 1961] Joachim Lambek. On the calculus of syntactic types. In *Structure of Language- and its Mathematical Aspects. Proceedings of the Symposia in Applied Mathematics, XII*, Providence, RI, 1961. American Mathematical Society.

[Pareschi and Steedman, 1987] Remo Pareschi and Mark Steedman. A lasy way to chart-parse with categorial grammars. In *Proceedings of the 25th Annual Meeting of the ACL*, pages 81–88, Stanford, CA, 1987.

[Schabes, 1990] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1990.

[Steedman, 1987] Mark Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5, 1987.