Combinatory Categorial Grammar – Introduction

A categorial grammar that includes only forward and backward functional application combinators is sometimes called a "**pure categorial grammar**." A pure categorial grammar, however, can be extended by including additional combinators. For example, a **combinatory categorial grammar** is obtained by including **functional composition** and **type-raising** combinators.

Type-raising

Type-raising can convert elementary syntactic types to functional types according to a rule such as the following:

(1) *Subject Type-raising*: (>**T**)
$NP \Rightarrow_{\mathsf{T}} S/(S\backslash NP)$

whereby the elementary NP type is turned into the functional $S/(S\backslash NP)$ type. This rule represents an instance of **forward type-raising** because the argument of the new function is to the right. Raising of this kind is also called **subject type-raising** because it is the subject NP that is raised to a functional type. The subscript '**T**' on the derivation arrow '⇒' identifies a type-raising operation, and the '>**T**' notation distinguishes type-raising from functional application in a derivation.

Example (2b) demonstrates the type-raising rule (1) in a derivation of the string 'birds fly.' For comparison, (2a) shows a derivation of the same string that does not employ type-raising.

(2) a.
$$\dfrac{\dfrac{\textsf{birds}}{NP} \quad \dfrac{\textsf{fly}}{S\backslash NP}}{S}\,<$$

b.
$$\dfrac{\dfrac{\dfrac{\text{birds}}{NP}}{S/(S\backslash NP)}\,>\mathsf{T} \quad \dfrac{\text{fly}}{S\backslash NP}}{S}\,>$$

The word 'birds' is associated with the elementary syntactic type NP in both (2a) and (2b), and 'fly' is associated with the functional type $S\backslash NP$. In (2a), backward application of the function $S\backslash NP$ to its argument on the left yields the result S. In (2b), the NP syntactic type of the subject 'birds' is converted by the forward type-raising rule in (1) to a function of the form $S/(S\backslash NP)$. Since the argument $S\backslash NP$ of the new function is the syntactic type of 'fly,' a forward functional application can be performed to yield the result S. Forward type-raising in (2b), followed by forward functional application, thus yields the same result as the backward functional application operation in (2a).

Type-raising rules such as (1) have been described as operations that "turn arguments into functions over functions-over-such arguments." In (2), the subject NP type is the argument of the functional $S\backslash NP$ type of the verb. Subject type-raising turns the NP argument into the $S/(S\backslash NP)$ type that is a function

over the function $S\backslash NP$ which takes the original NP as an argument. Type-raising of the NP in this case may thus be thought of as taking place in the context of the functional $S\backslash NP$ type of the next word in a sentence.

Alternatively, the functional type $S/(S\backslash NP)$ may be associated with the word 'birds' the lexicon as follows:

(3) birds := $S/(S\backslash NP)$

This association is consequently stored along with the association of 'birds' with the NP type, and perhaps N. According to this approach, rules such as (1) then just describe or represent a relationship or connection between the NP and $S/(S\backslash NP)$ syntactic types associated with 'birds' in the lexicon.

If raised types are associated with nouns in the lexicon, then type-raising does not operate during a derivation. Rather, the different types are retrieved from the lexicon. In a parallel processing model, the alternative type associations lead to concurrent alternative derivations. Or, in a serial processing model, heuristics might be employed according to which, if 'birds' for example is the first word in a string and is thus likely the subject of a sentence, the type $S/(S\backslash NP)$ is given precedence and is tried before the NP alternative.

Type-raising might be thought of as working something like the case-marking morphemes in languages such as Latin and Japanese. In these languages, the subject of a sentence is identified by a nominative case-marker. This morpheme is associated in the lexicon with a function that maps a noun into a function that takes a verb phrase as an argument. Languages such English and Chinese, however, lack case-marking morphemes. In these languages, it the position of a noun phrase relative to the verb that performs the role the case-markers. For example, the noun phrase that precedes the verb in a simple English sentence is normaly the subject, and implicitly carries nominative case. It is thus reasonable to treat nouns that can be subjects of sentences in English as being associated with a function over verb phrases such as in Example (3).

That the syntactic type $S/(S\backslash NP)$ associated with 'birds' in (3) is in fact a function over verb phrases can be made clear by substituting VP for the argument $S\backslash NP$. This substitution yields a function that can be written as S/VP. (Note that this function can be read as a claim that a noun phrase followed by a verb phrase will constitute a sentence.) If VP is also substituted for the functional type $S\backslash NP$ associated with the verb 'fly,' then application of the functional type S/VP associate with 'birds' to its argument VP associated with 'fly' yields the result S just as in the derivation in Example (2b).

The subject type-raising rule in Example (1) is a special case of the following general forward type-raising rule:

(4) *Forward Type-raising*: ($>\mathbf{T}$)
$X \Rightarrow_{\mathsf{T}} \mathrm{T}/(\mathrm{T}\backslash X)$

where T denotes a variable over syntactic types that gets instantiated during a derivation. X denotes an elementary syntatic type that is subject to the constraint that it be a complement of a verb. Since X corresponds to the

subject of a sentence, it is normally the NP type. With X being NP, and T instantiated with S, (4) reduces to Example (1).

There is also a backward analogue of forward type-raising that operates according to the following rule:

(5) *Backward Type-raising*: (<**T**)
$X \Rightarrow_{\mathsf{T}} \mathrm{T}\backslash(\mathrm{T}/X)$

T and X are subject to the same constraints as those that apply in (4); but in addition to NP, X may be a type such as S and PP that can be complements of verbs.

Backward type-raising of a **to**-PP, for example, can correpond to the association of dative case with an indirect object, while backward type-raising of an object NP corresponds to the association of accusative case with the NP. An example of object type-raising is given in the note on functional composition.

Functional Composition

A number of varieties of categorial grammar include type-raising. The composition of functional types, however, is often cited as the distinguishing charactistic of combinatory categorial grammars.

Two functional types can compose if the domain of one type corresponds to the range of the other. In the forward composition of two functions, the domain of the first function is the range of the second. The result of this composition is a new function with the range of the first function and the domain of the second. Forward functional composition operates according to the following rule schema:

(6) *Forward Composition*: (>**B**)
$Y/Z \;\; Z/X \;\Rightarrow_{\mathsf{B}}\; Y/X$

wherein each of the X, Y, and Z may be an elementary or a functional syntactic type. The functional form Y/X represents the syntactic type of the constituent formed by combining words or constituents with syntactic types represented by the functional forms Y/Z and Z/X. The '$\Rightarrow_{\mathsf{B}}$' and '>**B**' notation in (6) identify the functional composition operation in a derivation.

The following example illustrates forward functional composition in a derivation of the string 'birds like bugs:'

(7)
$$\dfrac{\dfrac{\dfrac{\textsf{birds}}{NP}}{\dfrac{S/(S\backslash NP)}{}}>\mathbf{T} \qquad \dfrac{\textsf{like}}{(S\backslash NP)/NP} \qquad \dfrac{\textsf{bugs}}{NP}}{\dfrac{S/NP}{}}>\mathbf{B}}{S}>$$

In the first step of this derivation, the NP type of 'birds' is raised by rule (1) as illustrated in Example (2b). The resulting function $S/(S\backslash NP)$ takes its

argument to the right. The next word of the string, 'like,' has the functional type $(S\backslash NP)/NP$. Since the domain $S\backslash NP$ of the first function $S/(S\backslash NP)$ is the range of the second function $(S\backslash NP)/NP$, the two functions can compose according to the following instance of the forward composition rule in (6):

(8) $S/(S\backslash NP)\ \ (S\backslash NP)/NP\ \Rightarrow_{\mathsf{B}}\ S/NP$

The two words 'birds' and 'like' consequently combine to form a constituent with the syntactic type S/NP. The word 'bugs' with syntactic type NP can then be combined with this constituent by forward functional application to complete the derivation. Figure 1. shows a tree representation of the derivation in Example (7).

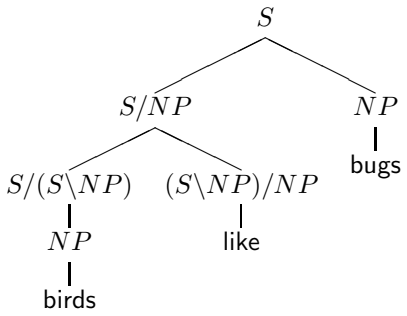

Figure 1: Tree representation of the derivation of the string 'birds like bugs' in Example (7).

This derivation can be compared with that in Figure 2. which shows a derivation of the same string but which does not employ type-raising and functional composition.

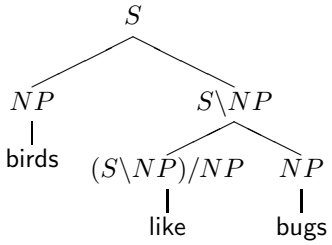

Figure 2: A derivation of the string 'birds like bugs' that does not employ functional composition and type-raising.

The comparison of these two figures reveals that the functional composition combinator permits the string to be derived incrementally. In an **incremental derivation** or **incremental syntactic analysis**, each word of a string is

incorporated into the derivation or analysis as soon as it is encountered while moving through the string from left to right. Each word is combined with the previous words to yield the syntactic type of a constituent that includes all the words encountered so far.

In Figure 1., the verb 'like' is combined with its subject NP 'birds' by functional composition to produce a constituent with the syntactic type S/NP. The object NP of the verb, 'bugs,' is then combined with the S/NP constituent by functional application to yield the syntactic type S. In contrast, the derivation represented by Figure 2. is not incremental: the verb 'like' cannot be combined with its subject 'birds' until after 'like' has been combined with its object 'bugs' by forward functional application. This operation yields the syntactic type $S\backslash NP$ that corresponds to a verb phrase constituent. Only then can the NP 'birds' be incorporated into the derivation by backward functional application.

The derivation in Figure 1. includes the constituent with type S/NP consisting of a noun phrase followed by a verb. Constituents such as this are not licensed by conventional context-free phrase-structure grammars. They have thus been described as "nonstandard constituents." These constituents are nonetheless "standard" in combinatory categorial grammar.

The S/NP constituent, for example, appears in derivations that include **object type-raising**. Object type-raising can be viewed as a special case of backward type-raising. If T in the backward type-raising rule schema (5) is instantiated by S, and X is NP, then the following rule is obtained:

(9) *Object Type-raising*: ($<\mathbf{T}$)
$NP \Rightarrow_{\mathsf{T}} \mathrm{S}\backslash(\mathrm{S}/NP)$

This rule, when employed to raise the NP type associated with the object 'bugs' of the verb 'like' yields the derivation of the string 'birds like bugs' shown in Figure 3.

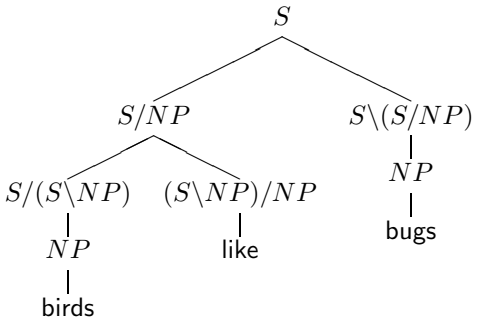

Figure 3: A derivation of the string 'birds like bugs' that demonstrates object type-raising.

The derivation of the string 'the cat gives dogs a chase' shown in Figure 4. illustrates two further examples of backward type raising.

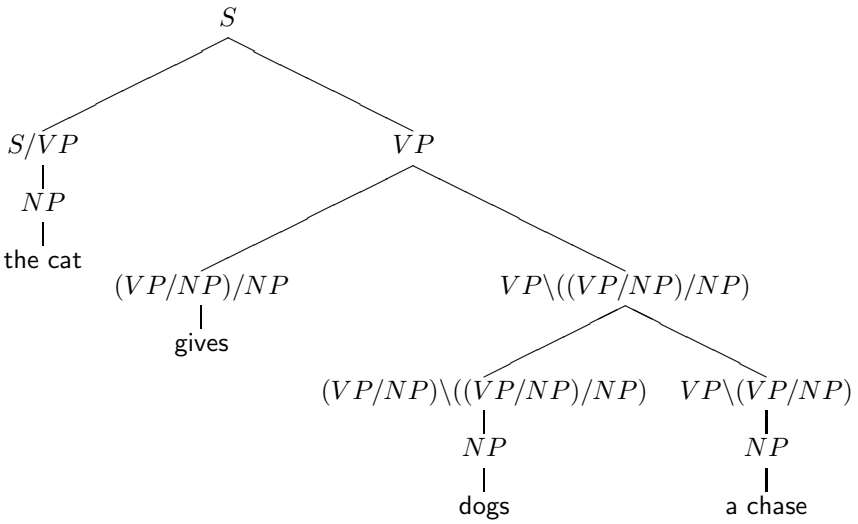

Figure 4: A derivation of the string'the cat gives dogs a chase' that demonstrates backward type-raising and backward functional composition.

In the backward raising of the NP type associated with 'dogs,' T in the schema (5) is instantiate by VP/NP, wherein VP stands for $S\backslash NP$. Thus we have

(10) $NP \Rightarrow_{\mathsf{T}} (VP/NP)\backslash((VP/NP)/NP)$

In the case of the NP type of the substring 'a chase,' T is instantiated as VP so that

(11) $NP \Rightarrow_{\mathsf{T}} VP\backslash(VP/NP)$

The $VP\backslash(VP/NP)$ and $(VP/NP)\backslash((VP/NP)/NP)$ types correspond to the association of accusative and dative case with the two NPs 'a chase' and 'dogs,' respectively.

Note that in Figure 4. the articles 'the' and 'a' in the substrings 'the cat' and 'a chase,' respectively, are associated with the syntactic type NP/N (like the adjective 'nice' illustrated previously). Forward functional application then yields the NP type in each case. Note also that 'gives' is associated with the syntactic type $(VP/NP)/NP$, where VP stands for $S\backslash NP$. The function $(VP/NP)/NP$ takes two NP arguments to the right, the first of which has dative case, and the second accusative.

The functional syntactic types the result from type-raising the two NPs reflect their syntactic case. These types can be combined by **backward functional composition** which operates according to the following rule schema:

(12) *Backward Composition*: ($<\mathbf{B}$)
$Z\backslash X \;\; Y\backslash Z \Rightarrow_{\mathsf{B}} \;\; Y\backslash X$

Z is the range of the first functional form, $Z\backslash X$, and the domain of the second, $Y\backslash Z$. In this example, VP/NP is substituted for Z. $(VP/NP)/NP$ is substituted for X, the argument of $Z\backslash X$, and VP for Y, the range of $Y\backslash Z$. With these substitutions, we have the following instance of the schema in (12).

(13) $(VP/NP)\backslash((VP/NP)/NP)\ \ VP\backslash(VP/NP)\ \Rightarrow_{\mathsf{B}}\ VP\backslash((VP/NP)/NP)$

Backward functional composition thus yields $VP\backslash((VP/NP)/NP)$ as the syntactic type of the argument cluster consisting of the two objects of the verb 'gives.'

This constituent can then be combined with the verb by backward functional application

(14) $(VP/NP)/NP)\ \ VP\backslash((VP/NP)/NP)\ \Rightarrow\ VP$

to yield the VP constituent of the sentence. The NP type of the subject 'the cat' is raised to the type S/VP to reflect its having nominative case. Forward functional application then yields the result S to complete the derivation.