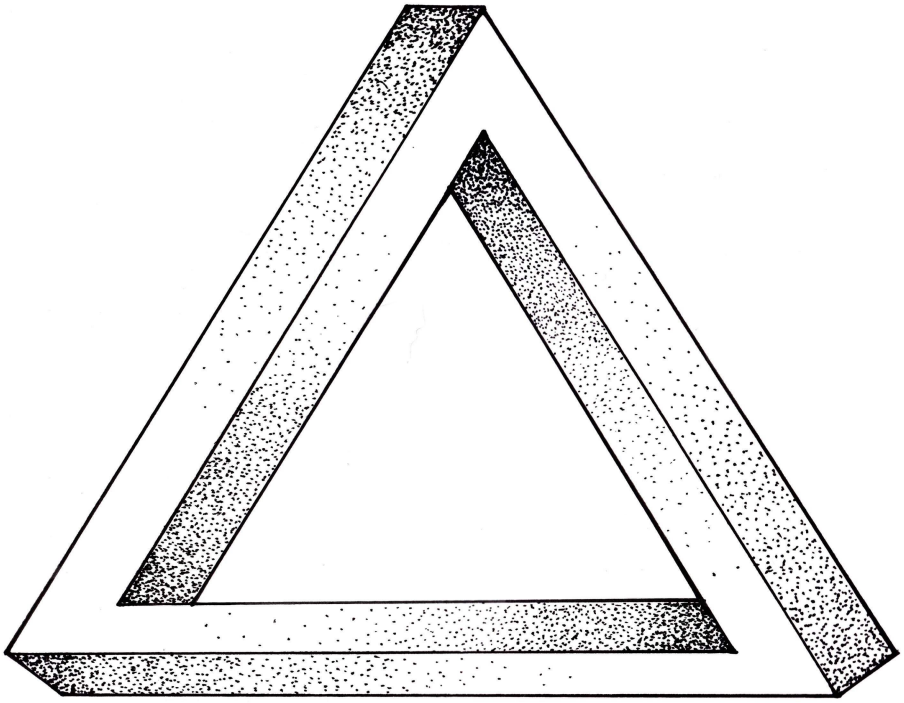
COGS542
Grammars, Combinators and Cognition
Lecture Notes

Tschomski

November 14, 2020



Contents

1 Introducing the Course

Combinators and their syntax-semantics. Combinators and the lexicon. Morphology and grammar. Parsing and language acquisition. Syntactic analyses of various phenomena using combinators. Codeterminism in syntax and semantics. Combinators and serializable knowledge.

1.1 Description of the Course

To gain an understanding of how order and its semantics can give rise to linguistic structure. To appreciate the different implications of type-dependent and structure-dependent grammars. To build models of syntax, morphology and parsing using combinators and Combinatory Categorial Grammar. To situate combinatory theories in cognitive science including philosophy, language, music, planning and reasoning. Third of the course is about combinators and structure of meaning. Another third is about computational language models with combinators, using CCG. The last third is about combinators and plan structure, which is exploratory. The course explores the idea that planning and language may have many things in common.

1.2 Syllabus

Week	Topic
1	Introducing the Course
2	Introduction to Combinators
3	Combinators in Natural Language
4	Combinators in Natural Language (cont'd)
5	Projection of Natural Language Combinators
6	Learning the projected component's items
7	Obtaining the projected component's items
8	Planning
9	Midterm Exam
10	Language of Thought
11	
12	Presentations
13	
14	

2 Introduction to Combinators

Basic idea behind combinators is to compositionally derive more interpretations of a sequence of elements which are themselves are meaningful. The underlined keywords are important notions in this definition in that what they are and what they mean. We need to look at these keywords in order to understand why combinators are essentially very useful understanding the cognition. Although he cognition is very parallel in signal transmission, when it comes to high level cognitive processes such as language, vision and planning, that is all massively serial. The reason why it is serial is that it requires a kind of synchronization and coordination. When we look at ourselves, the human beings, we see that we are sequence interpretation machines, we cannot help interpreting sequences without meaningful elements in the sequence.

2.1 What does it mean to be meaningful?

The order of the elements and the conventions in which they are used reveal the meaning of the sequence. Look at the following sequence.

$$(1) \cdots \text{John buy ticket fly} \cdots$$

This sequence can mean that John buys a ticket to fly because conventionally as the semantic properties of the elements require, one buys something since there must be a buyer and buyee, also this serves an aim. Therefore, we can extract the meaning above. Let's take a look at another sequence where the same elements are given yet in a different order.

$$(2) \cdots \text{John fly buy ticket} \cdots$$

In this sequence, you can derive the meaning that John wants to fly, so he buys a ticket. Depending on the order and interpretations, you begin to make different inferences for the outcome. The meaning of the sequence are updated when new conventions are included into the model. For example, considering the sequence (1), now suppose that the ticket is for a concert. In this case, the meaning that John buys a ticket for a concert and flies to where the concert takes place to use the ticket. The key point here is that different ordering of the elements leads to different interpretations and different conventions lead to different meanings despite the same order. This idea was put forward by Edmund Husserl in 1900. He said if you have a sensible distinction of meaning in a sequence, and only way you can discern that meaning difference and work with it is to have a different representation for it. This can be said to be the starting point of categorical grammar.

2.2 What are the combinators?

A combinator is a lambda term with no free variables. The reason why there is no free variable is that it gives you compositionality in a very rigorous way, and this is the first step understanding meaning of more and more complex sequences.

$$(3)\ \lambda x \lambda y.+\ x\ y \qquad (4)\ \lambda.+\ x\ y$$

The lambda term in (3) is a combinator because all the variables, x and y are bound, and there is not free variable[1]. However, the lambda term in (4) is not a combinator because it has a free variable, y. There is no binding for y; thus, it cannot be interpreted compositionally. Now let's take a look at the sequence $\cdots$ mother father $\cdots$. This sequence means that *"mother of father"*. Therefore, we can define two lambda terms as in (5) and (6).

$$(5)\ \text{mother} = \lambda x.mother\ x \qquad (6)\ \text{father} = \lambda x.father\ y$$

Using these lambda terms, we can find mother of father of someone as in (7). This is what we call function composition in mathematics.

$$(7)\ \lambda z.mother(father\ z)$$

However, there is a combinatory way of writing this without introducing any variables into the sequence. This is where we introduce combinators.

B $\stackrel{def}{=} \lambda f \lambda g \lambda x.f(gx)$ (compositor)

$\equiv \cdots$ **B** mother father $\cdots$

$\rightarrow$ In the lambda terms given above, the two terms are not related to each other, but when combinator **B** is used, the terms get a meaning together. This is the use of combinators.

S $\stackrel{def}{=} \lambda f \lambda g \lambda x.fx(gx)$ (substitutor)

$\cdots$ the books I have read without understanding $\cdots$

$(\lambda x.not - understand\ x(read\ x))books$

W $\stackrel{def}{=} \lambda f \lambda x.fxx$ (duplicator)

$\cdots$ John turn $\cdots$ *(Take the argument and copy it!)*

W.turn John $\rightarrow$ John turn John (himself)

C $\stackrel{def}{=} \lambda f \lambda g \lambda x.fxg$ (permutator)

[1]The symbol, + is a constant.

$\mathsf{I} \stackrel{def}{=} \lambda x.x$ (identificator)

$\cdots$ **C I** John turn $\cdots$ *(Take the sequence and change the order of the last two!)*

$\cdots$ **I** turn John $\cdots$ *(Take the argument and give it back as it is!)*

$\cdots$ turn John $\cdots$

$\mathsf{K} \stackrel{def}{=} \lambda f \lambda g.f$ (cancellator)

$\mathsf{T} \stackrel{def}{=} \lambda f \lambda g.gf$ (commutator)

QUIZ 1:

Question: Show that BSC=B(BW)B.

Hint: *Combinators are lambda terms; therefore, they apply the same way using function application (i.e. beta reduction). Recall that '=' is an extensional concept in this world. Two things are equal if they behave the same in arbitrary choice of elements. For example, M=N if and only if Ma = Na for arbitrary a.*
What all this means is that for example $Babc = a(bc)$, because $B = (\lambda f \lambda g \lambda x.f(gx))$, and $(\lambda f \lambda g \lambda x.(gx))abc = a(bc)$.
Another example: $BKabc = ab$ because $BKabc = K(ab)c = ab$.

Answer: Feeding both combinators with the same elements, x, y and z;

= BSCxyz

= S(Cx)yz

= (Cx)z(yz)

= x(yz)z

= B(BW)Bxyz

= (BW)(Bx)yz

= W((Bx)y)z

= ((Bx)y)zz

= x(yz)z

3 Combinators in Natural Language

Combinators are essentially mathematical objects, and the empirical domain of ontology for sequences you have such as sequences of actions and words etc. shows how combinatory theory for those will be dealing with their problems. Combinators tell what is possible and what is not in these empirical domains. Since the most worked out domain for combinators is natural language, combinators in natural language will be covered in this section. To introduce the notation in combinatory categorial grammar (CCG), let's see the following natural language string in (1).

(1) a. John likes Mary → Observable
b. NP (S\NP)/NP NP
c. $John'$ $\lambda x \lambda y.likes'xy$ $Mary'$

(1a) indicates the observables in other words the phonological elements on the surface. (1c) shows the predicate argument structure of *like*. Lambda terms in (1c) point out the prominence in that subjects are more prominent than direct objects, direct objects are more prominent than indirect objects and beneficiaries etc. Therefore, in natural language, arguments of predicates have relative prominence. The meanings of the observables come from the predicate argument structures of the lambda terms and the life-time experience on these elements. The issue here combinators do not tell anything about the order of the elements. To resolve this problem, some kind of a directionality can be put forward. In this directionality notation (/) means predicate needs an arguments to its right and (\) means predicate needs an argument to its left. In (1b), the phrase, S needs an argument to its left which is NP, John and to its right which is NP, Mary. The CCG notation for (1) is as follows:

(2) $\underset{PhonologicalElement}{likes' :=}$ $\underset{SyntacticType}{(\text{S}\backslash\text{NP})/\text{NP:}}$ $\underset{LogicalForm}{\lambda x \lambda y.likes'xy}$

This is how the sequences are represented showing their order and meaning. The drawing notation for this representation is as follows (FA means *Function Application*):

$$\dfrac{\dfrac{\underline{\text{John}} \quad \underline{\text{likes}} \quad \underline{\text{Mary}}}{\begin{array}{ccc}\text{NP} & (\text{S}\backslash\text{NP})/\text{NP} & \text{NP} \\ :John' & \lambda x \lambda y.likes'xy & :Mary'\end{array}}\ \text{FA}}{\begin{array}{c}\text{S}\backslash\text{NP} \\ :(\lambda x \lambda y.likes'xy)Mary' \\ = \lambda y.likes'Mary'y)\end{array}}\ \text{FA}$$

$$:(\lambda y.likes'Mary'y)John' = likes'\ Mary'\ John'$$

As can be seen having an argument on the right is function application and also having an argument to the left is function application as well.

$\alpha := X/Y : f\ \beta := Y : a \rightarrow \alpha\beta := X : fa$ (Forward Application)

$\alpha := Y : a\ \beta := X \setminus Y : f \rightarrow \alpha\beta := X : fa$ (Backward Application)

We can write as $X/Y\ Y \rightarrow X$ (>) and $Y\ X \setminus Y \rightarrow X$ (<).

These notations are the common usage in many CCG papers, and although it seems to do a sort of syntactic ordering, these rules do phonology, syntax and semantics all in a single computation, which the beauty of CCG.

The question here is why we need B and S combinators in CCG if all we do is function application. That is where things begin to get more interesting. Let's remember the lambda terms regarding the combinators B and S below:

$$B = \lambda f \lambda g \lambda x.f(gx) \qquad S = \lambda f \lambda g \lambda x.fx(gx)$$

Take the sequence $w_1w_2w_3$, their function composition and meanings. Through combinators, we get phonology, syntax and semantics of the elements all together. All functions are curried (schönfinkeled). Therefore, the function composition is going to look like below given in curried form.

Sequence:	w_1	w_2	w_3	$w_1w_2w_3$
(B)	X/Y	Y/Z	Z	$\rightarrow$ X
Meaning:	$: f$	$: g$	$: x$	$: f(gx)$

In curried form: $X/Y : f\ Y/Z : g \rightarrow X/Z : \lambda x.f(gx)$

Sequence:	w_1	w_2	w_3	$w_1w_2w_3$
(S)	(X/Y)/Z	Y/Z	Z	$\rightarrow$ X
Meaning:	$: f$	$: g$	$: x$	$: fx(gx)$

In curried form: $(X/Y)/Z : f\ Y/Z : g \rightarrow X/Z : \lambda x.fx(gx)$

Sequence:	w_1	w_2	w_1w_2
(T)	Y	X/Y	$\rightarrow$ X
Meaning:	$: a$	$: b$	$: ba$

In curried form: $Y : a\ X/Y : b \rightarrow X : ba$

3.1 Function Application

The use of combinators is different from function application. What is done by function application is to give the functions their arguments and incrementally process them. Let's, for example, rewrite the combinator, T by taking X/Y to the right in $X/Y : b$ $Y : a \rightarrow X : ba$.

$$Y : a \rightarrow X \setminus (X/Y) : \lambda p.pa$$

This process is called **type-raising**, and it is not the projective component unlike the combinators because it is not order-preserving. The notation given above is forward application as well. If you want to see backward application, it is going to look like as follows:

$$Y : a \rightarrow X \setminus Y : b \rightarrow X : ba$$

If we want to move the function, we are going to get the following:

$$Y : a \rightarrow X/(X \setminus Y) : \lambda p.pa$$

3.2 What is projected by the combinatory projection?

So far, it has been shown that the combinators deal with syntax, semantics and phonology at the same time in combinatory surface calculus. However, to appreciate their real power, we need to have some real empirical data to test all the theory put forward by CCG. For example, take the sentence, John likes Mary.

Example:	John	likes	Mary
	$NP : j'$	$(S \setminus NP)/NP$	$NP : m'$
		$\lambda x \lambda y.likes'xy$	

And when we do the function applications, we get:

$$= (\lambda x \lambda y.likes'xy)m'$$

$$= \lambda y.likes'm'y$$

$$= (\lambda y.likes'm'y)j'$$

$$= likes'm'j'$$

QUIZ 2:

Question: We can take the combinator W and try to symbolize it, just like I did for B and S. For simplicity, I will again assume that in its definition, $W = \lambda f \lambda x.fxx$, we are always looking forward in a sequence. So we get: $(X/Y)/Y : f\ Y : x \rightarrow X : fxx$

Why? Because f is a two argument function, and arguments are identical, hence $X/Y/Y$ rather than X/Y. Also, there is one input x to W, so we don't have $X/Y/Y\ Y\ Y \rightarrow X$, because that would mean three inputs to W.

After this introduction, try the same process for $C = \lambda x \lambda y \lambda z.xzy$ for the quiz. Note that W is already a two-input combinator, but C is not. So I suggest you first write the C as I did for B and S for all its inputs, then *"curry"* the result so that it creates a partial function, much like I did for B and S.

Answer: Unlike the combinator W, C takes two arguments, and neither of these arguments are a function in contrary to the combinator B and S. Therefore, we can have singular expressions like Y and Z (not like Y/Z) for the arguments as given below.

Sequence:	w_1	w_2	w_3	$w_1w_2w_3$
(C)	(X/Y)/Z	Y	Z	$\rightarrow$ X
Meaning:	$: x$	$: y$	$: z$	$: xzy$

In curried form: $(X/Y)/Z : x\ Y : y \rightarrow X/Z : \lambda z.xzy$

QUIZ 3:

Question: Take the following English expression, *the book that Mary quickly read.* Here "read" cannot be the transitive "read". It is a two-argument function. We are concerned about how to obtain the following combinations:

the book that[Mary[quickly read]].

By assigning categories to words in brackets, show how we can obtain the bracketed grouping using B and T. Method is to assign categories to three words in the brackets. Each category must contain a syntactic type and a lambda term, representing roughly its meaning. For example, we have $the := NP/N : \lambda x.def'x$ and $book := N : book'$. We are only interested in obtaining the bracketed string. You don't need to show how it combines with 'the', 'book', and 'that'.

Answer:

$$
\begin{array}{lccc}
\text{pho:} & \underline{\text{Mary}} & \underline{\text{quickly}} & \underline{\text{read}} \\
\text{syn:} & NP & Adv/VP & (VP \setminus NP)/NP \\
\text{sem:} & mary' & \lambda f.quickly' f & \lambda x \lambda y.read' xy
\end{array}
\quad FA
$$

$$
\frac{\begin{array}{c}(Adv \setminus NP)/NP \\ quickly'(\lambda x \lambda y.read' xy)\end{array}}{\begin{array}{c}Adv/(Adv \setminus NP) \\ \lambda p.p\ mary'\end{array}}\ T
$$

$$
\frac{}{\begin{array}{c}\lambda z(\lambda p.p\ mary')(quickly'(\lambda x \lambda y.read' xy)z) \\ = \lambda z(\lambda p.p\ mary')(quickly'(\lambda y.read' zy)) \\ = \lambda z(quickly'(\lambda y.read' zy)mary') \\ = \lambda z(quickly' read' z\ mary')\end{array}}\ B
$$

First, in the derivation, function application is done to form the main function which is *"quickly read"* because this is going to take the arguments (I have made this choice since *"quickly"* cannot stand alone and it acts as a single unit with *"read"*). Then, to merge *"Mary"* into the derivation, the combinator, T is used as the function is looking for something to its right first, but "Mary" is on the left. Thus, type-raising is applied here. Finally, since *"read"* is a transitive verb, yet here it seems to be intransitive (in fact it is not) as its argument happens to be somewhere else for now (We are not concerned with it right now). Therefore, to make its merge into the derivation, the combinator B is used (z will be replaced by *"the book"*). By this way, the expression *"the book that Mary quickly read"* is derived from its base form (?) *"Mary quickly read the book"*.

4 Projection of Natural Language Combinators

In natural language combinators, the directionality is of great importance.

$(> B)$ $X/Y : f \quad Y/Z : g \rightarrow X/Z : \lambda x.f(gx)$

$(> B_\times)$ $X/Y : f \quad Y \setminus Z : g \rightarrow X \setminus Z : \lambda x.f(gx)$ (forward crossing composition)

$(< B_\times)$ $Y/Z : g \quad X \setminus Y : f \rightarrow X/Z : \lambda x.f(gx)$ (backward crossing composition)

$(< B)$ $Y \setminus Z : g \quad X \setminus Y : f \rightarrow X \setminus Z : \lambda x.f(gx)$ (backward harmonic composition)

$(> S)$ $X/Y/Z : f \quad Y/Z : g \rightarrow X/Z : \lambda x.fx(gx)$

$(< S)$ $Y/Z : g \quad X/Y/Z : f \rightarrow X \setminus Z : \lambda x.fx(gx)$

$(> S_\times)$ $(X/Y) \setminus Z : f \quad Y \setminus Z : g \rightarrow X \setminus Z : \lambda x.fx(gx)$

$(< S_\times)$ $Y/Z : g \quad (X \setminus Y)/Z : f \rightarrow X/Z : \lambda x.fx(gx)$

$(> T)$ $X : a \rightarrow T/(T \setminus X) : \lambda p.pa$

$(< T)$ $X : a \rightarrow T \setminus (T/X) : \lambda p.pa$

4.1 Projection Principle of CCG

Everything in the lexicon is projected onto surface structures unchanged, undeleted and unmoved.

32.10dan devam et.