Merge: building syntactic structure

Introduction to Syntax, EGG 2011, Lecture 3

July 27th, 2011

1 A structured review

Our goal is to develop a theory of sentence structure, starting as simple as possible, and justifying every bit of added complexity. We started with:

Hypothesis 1 Sentences are strings of words.

- By itself, this makes no distinctions between different types of words.
- Thus we would have to write a new rule for every single sentence, listing each individual word that it is made up of.
- Or we could have one general rule that strings words together, but since we don't distinguish different types of words, this will derive mostly gibberish.

So then we moved on to:

Hypothesis 2 Sentences are strings of syntactic categories.

- We split up words into different categories.
- Then we write our rules in terms of these categories instead of specific words.
- This lets the rules be general enough to generate several similar sentences, but restricted enough to avoid generating the worst kinds of gibberish.

☞ But we still have to write rules that have a lot in common with each other without capturing repetitive patterns.

☞ In fact, we would need an infinite list of rules to get all possible sentence types.

This led us to:

Hypothesis 3 Sentences are hierarchical structures built up of syntactic categories.

- They are not flat strings at all, but built out of nested **constituents**.

- Some constituents form meaningful units that can show up in different places in different types of sentence. These special constituents are called **phrases**.
- Writing our rules in terms of phrases lets us make them extremely general and flexible.

Today we'll look at what kinds of rules and primitives we should adopt in our theory in order to build up the hierarchical, phrasal structures essential to syntax.

2 Top down or bottom up?

We need a rule or operation with the following properties:

- Creates hierarchical structure
- Derives and reflects the properties of constituency
- Is recursive

Now there are lots of specific kinds of rules and systems that we could propose that would have those properties.

☞ But we're not ready to start worrying about details, and we don't know enough yet to decide among lots of options.

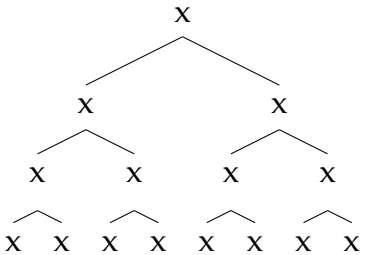
☞ What we want is something simple, general and abstract which we can then build on as we learn more.

The keys to keep in mind are recursivity and generality:

☞ Within a hierarchical structure, we find the same bits repeated over and over again if we look at an appropriate level of abstraction.

☞ We see a series of branching points, each representing where one bit of structure is made up of one or more bits of smaller structure

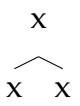
(1)



We just need a characterization of one such branching point.

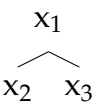
- If we make it appropriately general, it can apply to all such branching points.
- In that way, we can use it over and over again to characterize an entire hierarchical structure.
- I.e. we can analyze 1 as a series of instances of 2:

(2)



Now, there are in principle two main ways to describe a basic hierarchical structure like 2, repeated here as 3 with its parts labeled for ease of reference:

(3)



1. Top down: x_1 is made up of x_2 and x_3
2. Bottom up: x_2 and x_3 combine to form x_1

For most purposes, these two approaches will be essentially equivalent and yield the same results.

- So it's difficult to know a priori which to adopt for our theory and even whether the choice will end up having real consequences.
- But if we characterize the sentences of a language in terms of sequential derivations, then the direction in which derivations proceed does make a difference.
- It is extremely difficult to find clear empirical arguments to decide among the possibilities here – the argumentation operates mostly at a conceptual level.

We're going to adopt the hypothesis that a bottom up derivational approach models natural language syntax best without much argumentation. Here's one reason:

☞ The semantic interpretation of structures really has to be characterized in a bottom up fashion.

☞ If the syntactic derivation that creates the structures is also bottom up, we have the possibility of having syntax and semantics operate in parallel, step-by-step.

3 Merge

3.1 The fundamentals

Now we're ready to develop the operation which builds hierarchical structure bottom up. We start as simple as possible:

Merge: take a number of syntactic objects, and join them together to form a new syntactic object

(4) Merge X and Y to yield Z:

Some terminology:

Node: Object in a tree structure; Z, X and Y are nodes.

Branch Line connecting nodes

Mother: The node at the top of a branch, with respect to a node below; Z is the mother of X and Y

Daughter: The node at the bottom of a branch with respect to the node above; X and Y are daughters of Z

Sisters: Two nodes that have the same mother; two nodes that have been merged with each other; X and Y are sisters.

Root: The unique node in a tree that has no mother

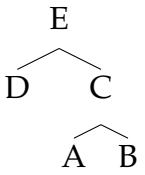
Terminal: A node that has no daughters

Note that Merge is **recursive**, as required:

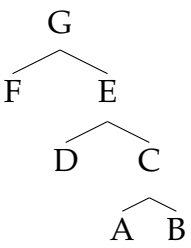
- ☞ It takes syntactic objects as its input and produces a new syntactic object as its output.
- ☞ That is, the output is the same type of thing as the input, and hence Merge can apply to its own output.
- ☞ So we can string together multiple instances of Merge to create ever larger structures.

(5) Merge A and B to form C:

(6) Merge D and C to form E:



(7) Merge E and F to form G:



⋮

Merge as we've defined it is completely general, and on its own it is unconstrained. The null hypothesis is that this is all there is to natural language syntax:

(8) **The simple Merge hypothesis:** Sentences are formed by successive applications of Merge, starting from the basic words of a language, and nothing else.

- Coupled with a complete lexicon, this will allow us to derive all of the sentences of a given language.
- And what it derives will be hierarchical structures that can accurately reflect the constituent structure of the sentences rather than flat strings.

Recall our example from last time:

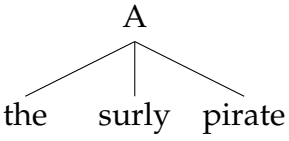
(9) The surly pirate drank the rum.

☞ Start with the following list of English words:

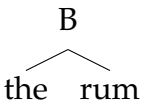
(10) {drank, pirate, rum, surly, the}

☞ Now we can build up the (still rather simple) constituent structure that we arrived at for this sentence by three successive applications of Merge.

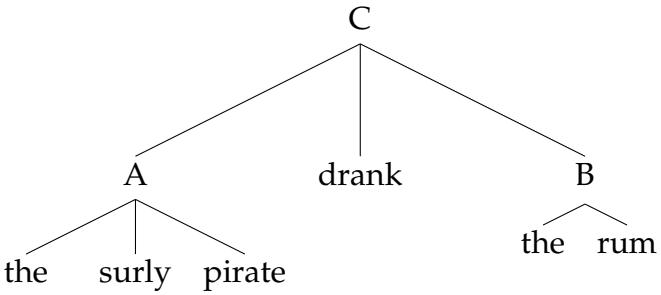
1. Merge *the*, *surly* and *pirate* to yield node A:



2. Merge *the* and *rum* to yield node B:



3. Merge A, *drank* and B to yield node C, the full sentence.

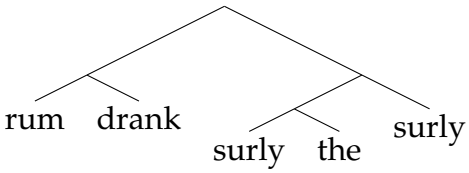


3.2 Formal restrictions

Of course, the theory of syntax embodied by 8 is far too powerful:

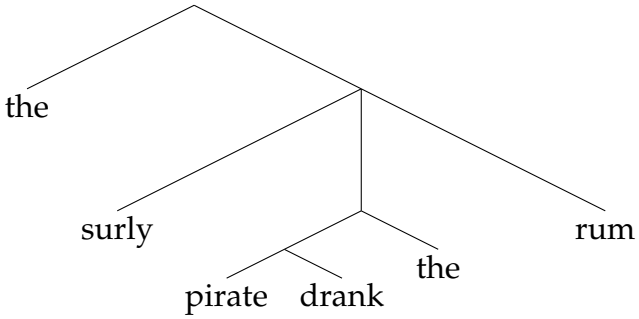
- It will happily derive every imaginable hierarchical structure composed of the words in our list.
- In addition to all of the actual sentences we want, we get all sorts of nonsense like 11:

(11)



- And we get things that look like real sentences, with the right words in the right order, but have the wrong structure:

(12)



So our job now is to figure out what restrictions to add to our simple hypothesis, constraining Merge so that it only gives us structures corresponding to grammatical sentences.

- ☞ Part of the problem, as you may have noticed, is that we haven't built sensitivity of syntactic categories into our system yet.
- ☞ But before we get to that, let's consider a couple of simple but powerful formal constraints on the operation of Merge.

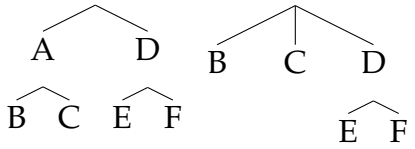
First, consider something that has been left implicit until now, but is absolutely crucial:

(13) **The Extension Condition** Merge always joins syntactic objects at their root nodes.

So the objects in 14a. and b. can only be merged as in 15a., not e.g., as in 15b.

(14) a. A b. D

(15) a. G b. A



Consider what this means:

- Merge takes whole constituents and combines them together on an equal footing.
- It can't take one constituent and put it inside another.
- The only way to get constituent Y inside constituent X is if X is the new constituent created by merging Y with something else.

Here's why this is needed:

- ☞ Without the Extension Condition, we could revise constituents in the course of the derivation.
- ☞ We could then no longer guarantee that the objects brought together by a single instance of Merge would ultimately form a constituent.
- ☞ This would make it extremely difficult to develop any principled account of what goes into constituency

Note that the Extension Condition also keeps things simple and clear in an important way:

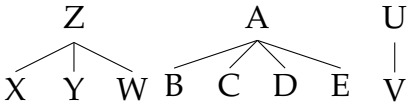
☞ You can always figure out the steps of a derivation just by looking at the structure that's output at the end.

Here's another formal restriction we should consider:

(16) **The Binary Branching Hypothesis** Merge always joins exactly **two** syntactic objects together, never more nor less.

So these guys are out:

(17)



This is a **working hypothesis**.

- There is nothing that would inherently restrict Merge to being binary, and none of its desirable properties that we've discussed would be lost if it weren't binary.
- However, binary Merge is the minimum operation necessary to build larger structures.
- So Occam's Razor dictates that we should try to get by with binary Merge alone, and only add more complicated operations when necessary.

So here's a more complete definition of Merge, updated to include the new constraints:

(18) **Merge:** Take two syntactic objects, and join them together at their roots to form a new syntactic object

4 Heads and projection

Before we can figure out how to further constrain Merge, there's a more basic issue we need to think about:

☞ Merge takes two (pre-existing) syntactic objects X and Y and puts them together to create a new syntactic object Z.

☞ We need to figure out what the properties of Z will be and make sure that Merge operates correctly to derive them.

☞ If Merge is to be general, simple and recursive, then clearly the properties of Z will be determined algorithmically on the basis of the properties of X and Y.

As usual, there are many ways we could imagine to do this, but a very simple and flexible approach is based on the idea of **projection** and **heads**:

- By default, the properties of Z are simply passed up or projected from either X or Y (perhaps with modifications).
- The object which projects its information up to the newly created mother node is called the head of the phrase.
- The head Is is the most important piece, the sub-part which determines the properties of the whole.

There are two levels at which we need to think about the identification of heads in order to work with them in our theory.

1. Given a chunk of actual linguistic structure, how do we figure out which part is functioning as the head? This is an empirical (or methodological) issue.
2. Given two syntactic objects which we are about to merge, how does our theory decide and indicate which will be the head? This is a theoretical issue.

We'll first work on the empirical issue before turning to the theoretical one.

Start from the observation that all words and phrases have restrictions on their syntactic distribution.

- E.g. *pigs* can be (informally speaking) the subject of *love*, or the object *eat*, but not the complement of *is*:

(19) Pigs love truffles.
(20) Humans love to eat pigs.
(21) * Peter is pigs.

Now, the strings in **boldface** in the following sentences are clearly constituents – e.g. they can all be replaced by *they*.

(22) **Those pigs** love truffles.
(23) **The old pigs** love truffles.
(24) **Some happy pigs which can fly** love truffles.
(25) **Some disgruntled old pigs in those ditches** love truffles.

Of course, they all involve the word *pigs*, and their meaning is clearly related to the meaning of *pigs*:

- *pigs* refers to porcine animals in general, or the whole set of such animals.
- These strings refer to more specific or restricted sets of porcine animals, not e.g. to ditches or flying.

Crucially then, the distributional restrictions on these strings are (mostly) the same as those on *pigs*:

(26) Humans love to eat **those pigs**
(27) Humans love to eat **the old pigs**
(28) Humans love to eat **some happy pigs which can fly**
(29) Humans love to eat **some disgruntled old pigs in those ditches**

(30) * Peter is **those pigs**
(31) * Peter is **the old pigs**
(32) * Peter is **some happy pigs which can fly**
(33) * Peter is **some disgruntled old pigs in those ditches**

➪ Clearly, *pigs* is determining the properties of these phrases, so it seems that *pigs* is the head in each case.

The head of a phrase typically has the following properties:

- It makes the central meaning contribution. The meaning of the phrase is a more specific version of the meaning of the head.
- It makes the central determination of the distributional restrictions on the phrase.
- It determines how the phrase is treated by syntactic and morphological processes like agreement.

Here's an example demonstrating that last point:

(34) The owners of the pigs love/*loves truffles.
(35) The owner of the pig *love/loves truffles.
(36) The owners of the pig love/*loves truffles.
(37) The owner of the pigs *love/loves truffles.

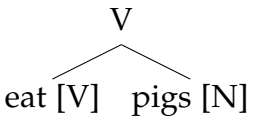
☞ Verb agreement here tracks the number of *owner/owners*, not *pig/pigs*, so the former must be the head.

☞ Note that this is corroborated by the fact that *the owner of the pig* is a kind of ower, not a kind of pig.

So how do we deal with the theoretical aspect of head identification?

☞ Remember, we're going to assume that being the head means that your properties project up to your mother node, while those of your (non-head) sister do not.

☞ Simplifying quite a bit, we can represent this in terms of syntactic category information, like this:

(38)


- *eat* is a verb, *pigs* is a noun, and *eat* is the head of the phrase *eat pigs*, so the category of the whole phrase is V.
- This captures the fact that *eat pigs* has a lot of the same distribution restrictions as a simple verb like *arrive*, not those of the noun *pigs*.

But we're left we a number of questions:

? How does our theory actually determine that it is the V from *eat* that should project here and not the N from *pigs*, i.e. how does it decide that *eat* should be the head?

? What other properties do syntactic objects have been categories, and how does the theory deal with them?

? How does the theory make sure that we even have phrases like *eat pigs* but not phrases like *snore pigs*?

In order to go after these issues, we need to step back and deal with something even more than Merge.

5 Introducing Features

What are the basic pieces that syntax deals with?

- We might think syntax just takes words and puts them together to form sentences. We've sort of been acting as though that were the case.
- But it turns out that syntactic processes can be sensitive to smaller bits, subparts of words.

(39) This potato tastes funny.
(40) These potatoes taste funny.
(41) * These potatoes tastes funny.
(42) * This potato taste funny.

- *this* and *tastes* only go together with *potato*, while *potatoes* requires *these* and *taste*.
- In a sense, *potato* and *potatoes* are the same word, but the syntax is clearly sensitive to the difference between them.
- We say that *potato* is singular, while *potatoes* is plural – two different **word-forms** of a single **lexeme**.

So *potato* and *potatoes* are mostly the same, but they differ in this one point of number.

☞ We need a theoretical concept for properties like number that constitute potential differences between words.

☞ Following standard linguistic practice, we will call them **features**, and they will play a big role in our theory.

A feature encodes a property of a word or other syntactic unit.

- Just like people have properties like their name, age and height, syntactic objects have properties, and we can represent these with features.
- An individual feature should correspond to a minimal possible distinction between two words.
- So if two words are exactly the same except for one property which can't be further broken down, they should differ in terms of exactly one feature.

Note that by adopting a notion of features, we are adding something non-trivial to our theory, so we want to be sure that it's justified and do it in the simplest way possible.

☞ The justification is the assumption that syntactic objects have properties which distinguish among them beyond the structural configurations in which they occur.[1]

[1] We should note that while this assumption may seem innocuous and well-motivated, it is not entirely uncontroversial. Boeckx e.g. has argued in recent work for the elimination of features from the narrow syntax, with all non-structural differences between syntactic terminals being a matter of the interfaces.

☞ Features in their most basic form attempt to encode this assumption as minimally as possible, and nothing more.

Features are normally written inside square brackets, separated by commas:

(43) [masculine, singular, nominative]

☞ This might be used to refer to English *he*, which simultaneously has all three of those features.

☞ They distinguish *he* from e.g. *she*, *they* and *him*.

We use features to describe all sorts of properties of words, and also to talk about how they interact with each other.

☞ So the fact that *potatoes* is [plural] tells us something about its meaning – that it refers to more than one entity.

☞ But it also conditions the way that the rules of adjective and verb agreement apply.

One kind of feature that we'll certainly need to assume is syntactic category.

- We've already seen that syntactic categories play an important role in generalizations about the kinds of syntactic structures that are possible.
- There are many differences between *pig* and *eat*, and many of them can be handled if we assume that the two words differ in terms of their categorial features.
- Exactly how many distinct categories there are and what kinds of features we should use to label them is going to be matter for us to resolve empirically.
- For now we can start with the four traditional categories noun, verb, adjective and preposition, and indicate them with the privative features N, V, A, P.

Now, features in the sense we are using them are an extremely general and flexible device. I.e. they are very powerful.

☞ So if we add them to our theory, we should make full use of that power and avoid positing additional theoretical objects in cases where features could be used instead.

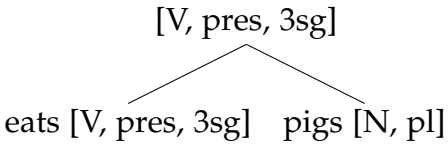
- ☞ Our desideratum should be to encode everything about words and larger syntactic objects which are not encoded by hierarchical structure.
- ☞ If we can use features to encode everything, then a word is actually nothing more than a bundle of particular features.

(44) **The exhaustive feature hypothesis:** The syntactic structure of a sentence consists of the hierarchical arrangement of bundles of features and nothing else.

Now we're ready to come back to the issue of projection. Given the assumption of features, we can say the following:

- Upon Merge, all of the features of the head, including its category feature, will be default project up to the newly created mother node.
- Looking again at the combination of a verb with its direct object, we can assume that not only the V feature, but also the tense and agreement features of the verb will project:

(45)



This gives us a way to **indicate** headedness in our structures, but we still don't have a way for our theory to predict it.

- ☞ I.e. there's nothing yet about the definition of Merge or the feature specifications on *eats* and *pigs* that determines that *eats* will be the head when the two Merge.
- ☞ This will be one of our tasks for next time.