Constraining Merge: Selection and dependent features

Introduction to Syntax, Topic 6

Sandhya Sundaresan, EGG Summer School, Wrocław 2019

August 5, 2019

Predicates have more to say about their arguments than just their thematic roles:

(1) Sue felt a feverish sensation.

(2) Sue felt feverish.

(3) Sue felt that she had a fever.

- ☞ feel takes an EXPERIENCER subject and a THEME object.
- ☞ But the object can be a noun phrase, an adjective or an entire clause.

Other predicates are more particular:

(4) Sue perceived a feverish sensation.
(5) * Sue perceived feverish.
(6) Sue perceived that she had a fever.

(7) # Sue became a feverish sensation.
(8) Sue became feverish.
(9) * Sue became that she had a fever.

(10) * Sue thought a feverish sensation.
(11) * Sue thought feverish.
(12) Sue thought that she had a fever.

The term we use for this phenomenon is c-selection (short for category selection) or subcategorization.

- C-selection seems to be a truly syntactic matter, since it cannot be derived from the semantic properties of predicates and their arguments.
- So when we describe the argument-taking properties of a given predicate in our theory, we'll need to include several different kinds of information.

Dependent features

How do we encode c-selection in our grammar?

- We want to represent everything in terms of features, and c-selection should be no different.
- But the c-selectional properties of a given lexical item don't tell us about a property directly observable on the item itself.
- Rather, they say something about how it fits into a sentence, what sorts of things it can or must combine with.

➪ We need a special kind of feature for this.

We need to distinguish independent features from the new kind we'll need, the dependent features:

Independent features like number on a noun or tense on a verb give us information about properties of the word itself, typically ones that have a clear meaning and/or an effect on the form.

Dependent features give information about the contexts in which syntactic objects can occur. They aren't associated with any particular form or meaning of the objects themselves, but act as instructions for putting sentences together in the right way.

We will mark dependent features with a u in front of them (more on this notation in a moment):

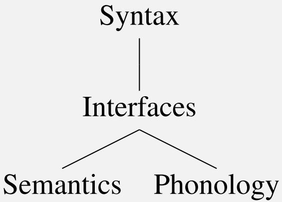
(13) X [G, uF]

Now we need to set things up so that dependent features can do some work. Here's the first step:

(14) The Principle of Full Interpretation
The structure output by the syntax may not contain any dependent features.

- This is nothing more than a formal statement about features that are meant to trigger syntactic operations.
- The real motivation is simply the assumption that we want to use features to implement syntactic requirements.

A conceptual underpinning for this is often assumed, but it is not strictly necessary or directly motivated by what we've seen. The idea is based on assuming the following for the derivation:



- The syntax creates structures and then sends them off to the two interfaces.
- The semantic interface determines a meaning for the structure, while the phonological interface determines a pronuncation.

On top of this, we could assume the following:

- Dependent features are something that the syntax knows how to deal with, but they are not legitimate objects for the semantics.
- Because of this, they are not even allowed to be around when the semantics goes to work.
- Therefore the derivation actually needs to eliminate them before the syntax finishes its work and hands things over to the semantics.

Whatever the motivation, we need a way to get rid of dependent features in the syntax, before they get to the interfaces.

(15) The Checking Requirement
Dependent features must be checked, and once checked, they can delete.

(16) Checking under Sisterhood
A dependent feature F on a syntactic object Y is checked when Y is sister to another syntactic object Z which bears a matching feature F.

- E.g. Y in 17 has a dependent feature [uF], so if left unchecked it would lead to ungrammaticality.
- We can merge Y with Z, which has a matching independent feature F.
- Now Y and Z are sisters, so [uF] can be checked off, and everything turns out ok.

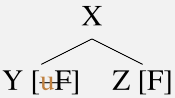
(17) Y [uF]

(18)

- E.g. Y in 17 has a dependent feature [uF], so if left unchecked it would lead to ungrammaticality.
- We can merge Y with Z, which has a matching independent feature F.
- Now Y and Z are sisters, so [uF] can be checked off, and everything turns out ok.

(17) Y [uF]

(18)



What's the point of all this? It becomes important when we consider how linguistics pieces Merge with one other to create syntactic constituents.

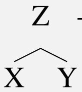
Before I talk about this, let's take a quick look at what Merge formally means.

Introducing Merge

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

(19) Merge X and Y to yield Z:



–or–

$[_Z$ X Y $]$

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.

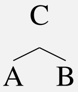
Root: The unique node in a tree that has no mother

Terminal: A node that has no daughters

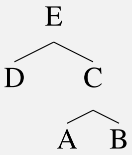
Note that Merge is recursive, as required:

- ☞ 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.

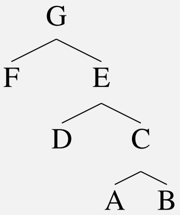
(20) Merge A and B to form C:



(21) Merge D and C to form E:



(22) 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:

(23) The "Only 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.

Consider this example:

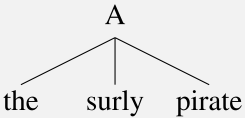
(24) The surly pirate drank the rum.

☞ Start with the following list of English words:

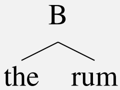
(25) {drank, pirate, rum, surly, the}

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

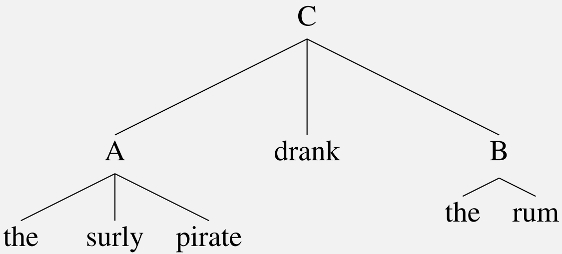
1. Merge the, surly and pirate to yield node A:



2. Merge the and rum to yield node B:



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

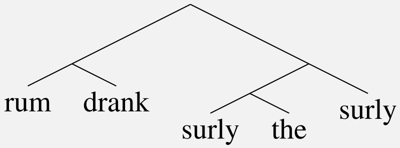


Formal restrictions

Of course, the theory of syntax embodied by 23 is far too powerful:

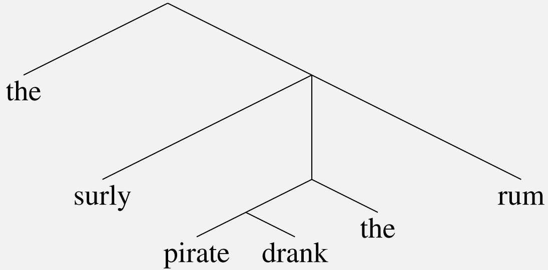
- It will happily derive every imaginable hierarchical structure composed of the words in our list.
- In addition to all of the actual sentences we want, we get all sorts of nonsense like 26:

(26)



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

(27)



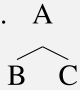
So our job now is to figure out what restrictions to add to our simple hypothesis, constraining Merge so that it only gives us structures corresponding to grammatical sentences.

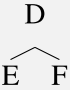
- ☞ Part of the problem, as you may have noticed, is that we haven't built sensitivity to syntactic categories into our system yet.
- ☞ But before we get to that, let's consider a couple of simple but powerful formal constraints on the operation of Merge.

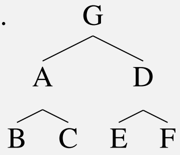
First, consider something that has been left implicit until now, but is absolutely crucial:

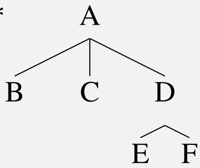
(28) The Extension Condition
Merge always joins syntactic objects at their root nodes.

So the objects in 29a and b can only be merged as in 30a, not e.g., as in 30b.

(29) a.


b.


(30) a.


b. *


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:

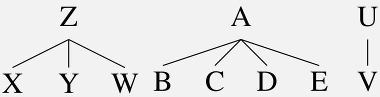
- ☞ You can always figure out the steps of a derivation just by looking at the structure that's output at the end.

Here's another formal restriction we should consider:

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

So these guys are out:

(32)



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:

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

Now we are in a position to see why a theory of c-selection is needed for Merge:

- ☞ Essentially we can use the dependent features in c-selection as instructions, triggers for appropriate instantiations of Merge.
- ☞ If a syntactic object doesn't Merge with the sort of thing demanded by its dependent features, the derivation will crash, i.e. it will fail to derive a grammatical sentence.
- ☞ This is how we can ensure that only those derivations succeed in which the right sort of things have Merged.

Specifically, we can now encode c-selection with dependent category features:

(34) kiss [V, uN]

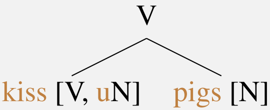
(35)

(36)

Specifically, we can now encode c-selection with dependent category features:

(34) kiss [V, uN]

(35)

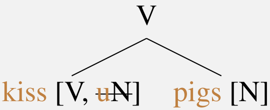


(36)

Specifically, we can now encode c-selection with dependent category features:

(34) kiss [V, uN]

(35)

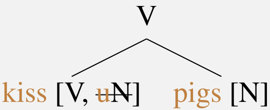


(36)

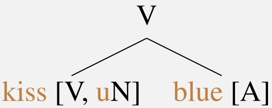
Specifically, we can now encode c-selection with dependent category features:

(34) kiss [V, uN]

(35)



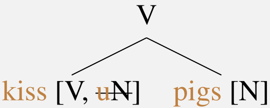
(36)



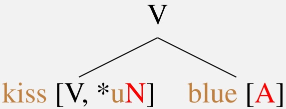
Specifically, we can now encode c-selection with dependent category features:

(34) kiss [V, uN]

(35)



(36)



- So our theory can correctly rule out sentences where the wrong category of argument combines with a predicate.
- It can also rule out sentences where a predicate doesn't combine with enough arguments.
- Either way, an unchecked dependent category feature will be left over at the end, causing a crash.

In addition to c-selection, we also need s-selection.

- This is where we encode the requirements a predicate places on the semantic type of its arguments.
- E.g. the object of ask can be of various syntactic categories, but it has to be semantically a question or piece of information that can be queried.

We won't really worry about s-selection, but you should know that it exists and seems to be independent of c-selection.

For purposes of comparison with other Minimalist theories (including the one in David Adger's 2003 book Core Syntax), note the following:

- The distinction that we are drawing between dependent and independent corresponds essentially to what those theories call uninterpretable and interpretable.
- This is the explanation for the u notation we are using for dependent features.
- I am not adopting this terminology here because it is tied to a particular set of assumptions about the status of these features which we cannot motivate.
- In our insistence on simplicity and generality, we will also depart from the standard theory of interpretability of features in other ways as we move forward.
- But most of the insights we develop here will be easily translatable into such a theory.

Back to heads

We can bring this all together to model the determination of the head in a given phrase:

(37) Definition of Head
The head of a phrase is the syntactic object which selects the other object which it Merges with to create the phrase.

➪ So the object that has a dependent category feature checked off in the Merge process is the head.

And we can set down the importance of being the head:

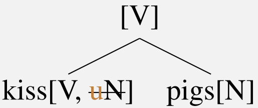
(38) **Headedness**
The item that selects is the item that projects.

- Imagine that object X selects object Y, merging with it to create object Z.
- The further properties of object Z will be projected from the head, object X.

An example:

- The constituent kiss pigs is headed by kiss, because kiss selects a noun like pigs.

(39)



- So kiss pigs is essentially verbal, as kiss is verbal, and has a distribution related to verbs, not nouns:

(40) a. I want to [$_V$ sing].
     b. I want to [kiss pigs].

(41) a. I want [$_N$ pigs].
     b. * I want [kiss pigs].

The way things are set up lets us derive an interesting corollary:

(42) Ban on Unchecked Features on Non-heads
If X selects Y and the two Merge, Y cannot have any unchecked dependent features.

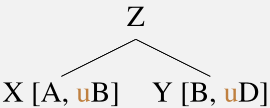
- In other words, only the head can have unchecked features.

Consider why this is:

- When X and Y Merge, the features from X will project to the newly created constituent, but the features of Y won't.
- When this merges with something else, the features projected from X can be checked, but those on Y can't, because Y won't be the sister of the newly merged object.
- Any dependent features on Y will thus remain forever unchecked, leading to a crash.

In structural terms:

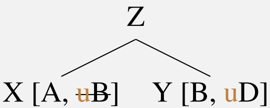
(43)



(44)

In structural terms:

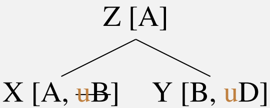
(43)



(44)

In structural terms:

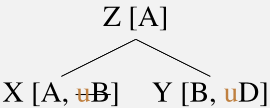
(43)



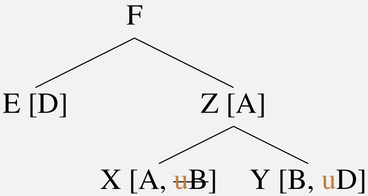
(44)

In structural terms:

(43)

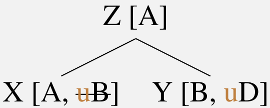


(44)

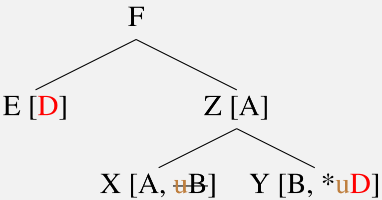


In structural terms:

(43)



(44)

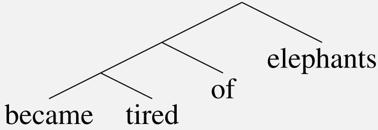


There is evidence that this is actually correct. Consider:

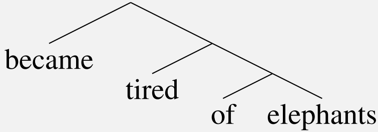
(45) Ellie became tired of elephants.

- The verb become c-selects for an adjective, and the adjective tired c-selects in turn for a preposition, and the preposition of c-selects for a noun.
- 45 has all the right things for those requirements to be satisfied, but we could imagine them being combined lots of different ways.

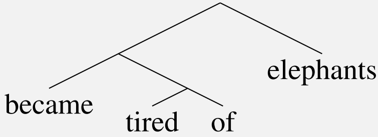
(46)


became
tired
of
elephants

(47)


became
tired
of
elephants

(48)


became
tired
of
elephants

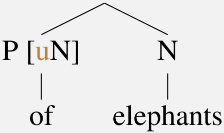
⋮

But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)
```
     N
     |
 elephants
```

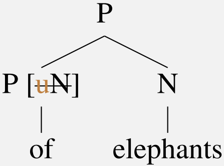
But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)



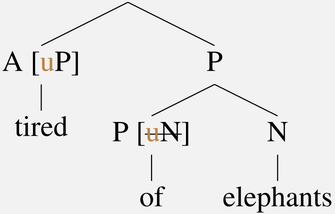
But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)



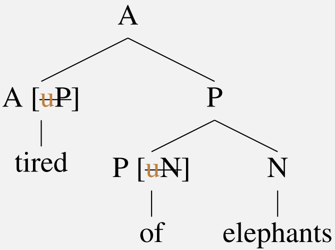
But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)



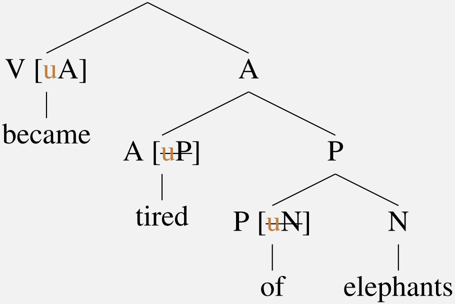
But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)



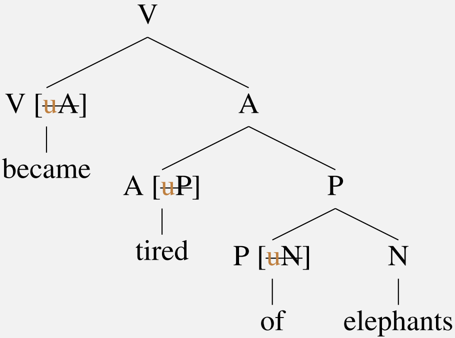
But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)



But the ban in 42 predicts that only one structure is possible: the one where the selectional feature on each object is checked before it itself is selected:

(49)



This is a good result, because constituency tests pick out the same structure. E.g.:

(50) [Tired of elephants] is something Ellie will never become.

(51) * [Become tired] is something Ellie never will of elephants.

Consider:

(52) * letters to

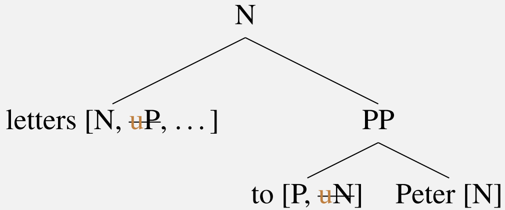
(53) letters [to Peter]

- to by itself is lacking something. It selects for an N but hasn't combined with one yet, so Merging it with letters is ungrammatical.
- But to Peter is complete, the [uN] selection feature on to having been checked, so it can Merge with letters.

Constituents like to Peter, which have checked all their dependent features, are called maximal objects or phrases.

- A maximal object built around a noun is an NP, one built around a P is a PP etc.

(54)



Being Maximal depends on having no unchecked dependent features.

- ➪ So there's nothing to stop something from being simultaneously Maximal and Minimal
- ☞ A simple lexical item with no selectional features, like Peter, will be both at the same time.

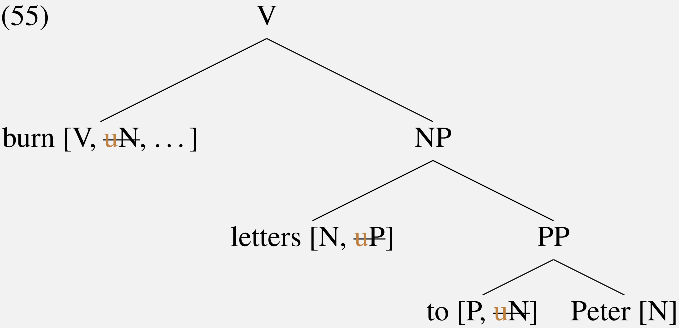
Note also that labeling a constituent as a PP or NP is just helpful notation and has no theoretical significance.

- ☞ The fact that an object is maximal is determined by its feature specification and nothing else.

Complements

A particular kind of structure arises when we Merge a simple lexical item with a category that it selects:

(55)



Peter is the complement of to, PP the complement of letters, NP the complement of burn. . .

The complement is the first thing selected by a head which Merges with that head.

- ☞ Note that being a complement has nothing directly to do with linear order.
- ☞ So in many languages, complements come before heads:

(56) Hanako ga Taro o tataku (Japanese)
Hanako nom Taro acc hit
'Hanako is hitting Taro.'

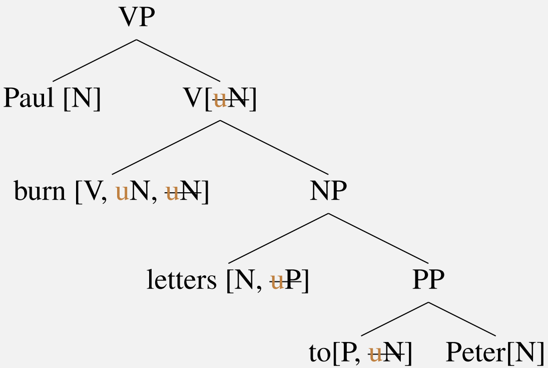
We unfortunatley won't get a chance to talk in detail about how to deal with differences like this, but it's important to note that it exists.

Specifiers

Something different happens when we add the subject:

(57) Paul burns letters to Peter.

(58)



- Paul is selected here by one of the [uN] features on burn. But it doesn't Merge directly with burn.
- Instead, it Merges with a higher projection, after burn has already Merged with leters to Peter.
- The thing that Paul merges with is neither maximal nor minimal. We'll call it an intermediate projection, which we sometimes indicate as $\bar{X}$ or X', pronounced X-bar.

☞ Something which is selected by and Merges with an $\bar{X}$ level projection is called a specifier.