Lecture notes: Week 3
Merge-based syntax

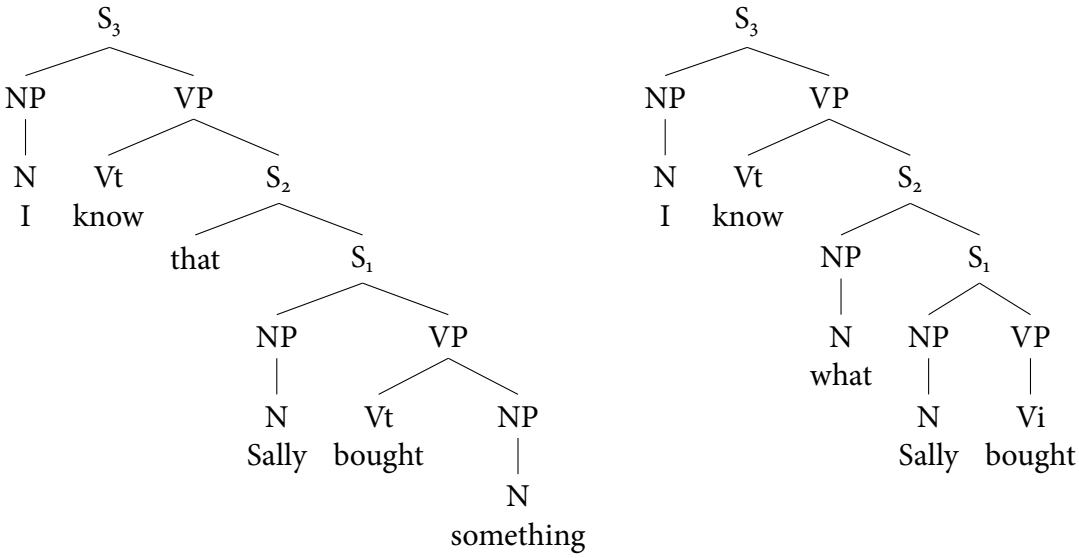
1 Back to displacement

- So far, we have seen that sentences like this are problematic for our view of Phrase Structure Rules:

  (1) a. I know that Sally bought something.
  b. I know what Sally bought.

- Although our phrase structure grammar can generate trees for them, it requires that we assume that *bought* is an optionally intransitive verb.

(2)



- This seems problematic. **Sally bought all day yesterday* is ungrammatical unlike real optionally intransitive verbs *Sally shopped all day yesterday*.

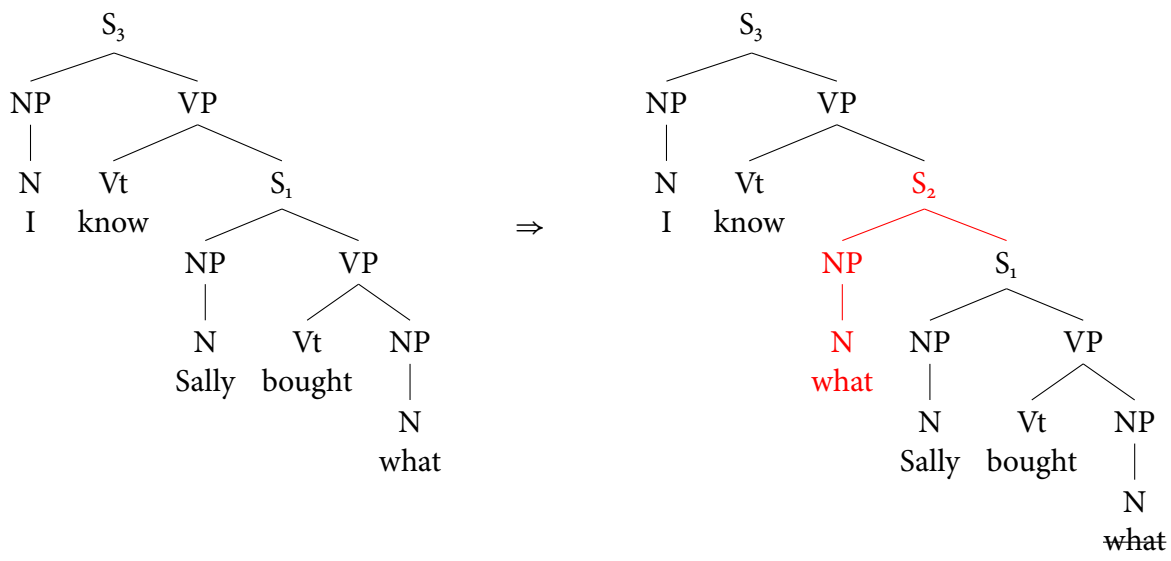
- It seems what we really want to say is that *buy/bought* is always a transitive verb, i.e. it always requires a subject and an object.

- In fact, you might have the intuition that *what* differs from *that* in the sentences above in being interpreted as the object of the verb.

- So, it seems we want to say that *bought* is in fact always a transitive verb. In order to account for this, we can appeal to the process of displacement that we looked at last time.

- On this view, *what* begins life as the object of *bought* (allowing us to stick with the transitive VP rule and assign it to category V_t) and then subsequently it is displaced to another position in the structure:

(3)



- What kind of rule is this? It is not a phrase structure rule. As (3) indicates, it is a kind of mapping between trees.
- What we want is an operation that applies to the trees that were already created by our PSRs. What kind of operation could that be?
- Before we proceed in answering this, let's take a moment to get some terminology under our belts.

Dominance

a. X dominates Y if it is possible to trace a strictly downward path through the tree to Y.

b. X is immediately dominates of Y if it is possible to trace a strictly downward path through the tree to Y and there is no Z such that Z dominates Y and X dominates Z.

- This allows us to define some other relations between nodes in the tree:

Mother

X is a(n) (immediate) mother of Y if X (immediately) dominates Y.

Daughter

X is a(n) (immediate) daughter of Y if Y (immediately) dominates X.

Sister

X is a sister of Y if X and Y share an immediate mother.

- So in (3) S_3 has the immediate daughters NP and VP. NP and VP are sisters since they share a common mother, S_3.

- All nodes in a tree are daughters of the root node (S_3).
- S_2 is a daughter, but not immediate daughter, of S_3 because there is another node (VP) that dominates S_2 and S_3 dominates S_2.
- Now that we have this terminology in place, let's return to the kind of operation that can derive the red part of the tree in (3).

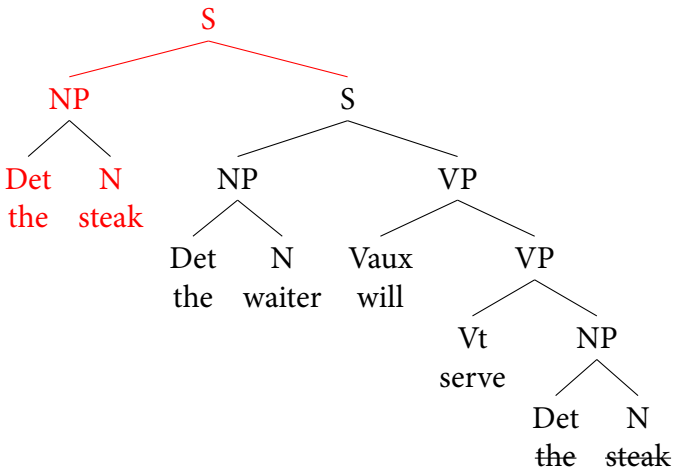
Move

Create a new node Z with immediate daughters X and Y.

Conditions:
a. X is a daughter of Y.
b. Do not pronounce X in Y.
c. If X = NP and Y = S, then Z = S.

- In (3), we create a new node S_2 (Z) with daughters S_1 (Y) and the NP *what* (X). We can do this because the NP *what* (X) is a daughter of S_1 (Y).
- Now we have a general rule of displacement (for NPs at least).
- It applies the sentences we saw last time with our constituency tests:

(4) The steak, the waiter will serve ___ (soon). The fish, he won't.



- Let's take stock of where we are now. In order to have a restrictive theory of everything we have talked about so far, we need both Phrase Structure Rules (with subcategories of V) to build up our trees and then this new operation *Move* that applies to an already assembled tree.
- At this point, we might want to ask ourselves whether we actually need PSRs any more. The *Move* operation is one that adds a new branching node to a previously assembled tree.

- We now hypothesized two distinct ways of building up tree structures. This is conceptually redundant if we can show that either one of the two ways can do everything on its own (*Occam's Razor*).
- We have seen that PSRs cannot handle displacement easily, but can a more general version of *Move* do the job of PSRs?
- The answer, I think, is yes. Let's reformulate *Move* into a more general operation *Merge*:

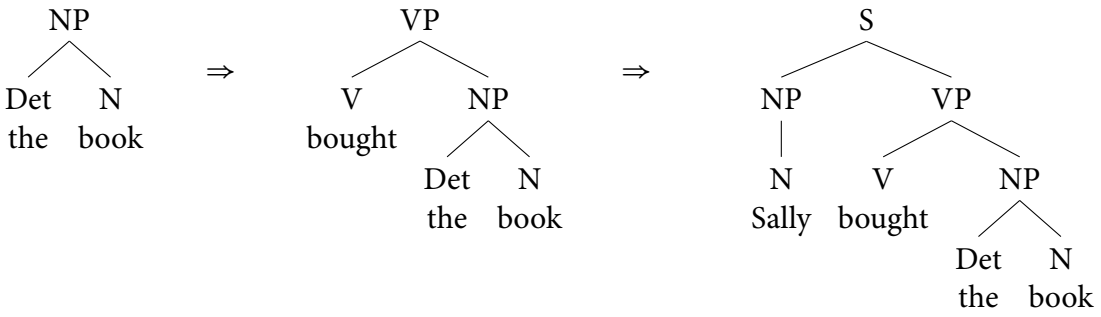
Merge

Create a new node Z with immediate daughters X and Y.

Conditions:
a. If X is a daughter of Y, do not pronounce X in Y.
b. If X = NP and Y = S, then Z = S.
c. If X = NP and Y = VP, then Z = S
d. If X = V and Y = NP, then Z = VP.
…

- *Merge* is simply an operation that puts two things together by creating a shared mother for them.
- Funnily enough, since Merge stipulates that branching nodes have two daughters, there are certain structures we cannot derive such as a phrase with one or more than two daughters. We will come back to this.
- We still need to add conditions. For the case of displacement, the condition in (a) is now re-phrased as a conditional statement, rather than an absolute condition on the operation.
- As with *Move*, *Merge* creates a series of mappings between trees:

(5)



- This sequence of steps is what we call a syntactic **derivation**.
- As we see above we still need to somehow state, just like PSRs, what the outcome of each possible combination of phrases is, e.g. that the outcome of merging V and NP is VP (and not NP, or something else).
- While listing all possible combinations is still an option, shifting to this view of structure-building by Merge offers a more general way to determine the outcome of putting two things together.

2 Building trees with Merge

- The Merge-based view of syntax involves somewhat of a conceptual shift. We no longer think of the trees as the scaffolding with words later added to them. On the Merge-based view, the words are the basic units that drive how trees are put together (not the rules).
- So, we can think of words as being stored in a mental **lexicon**. This is just a list of all the words in the language and their relevant properties (syntactic, phonological, semantic, and others).
- The things listed in the lexicon can be referred to as **lexical items**.
- These properties can be encoded as **features**.
- The view of features we will adopt here is one in which they have two parts: an **attribute** (this says what kind of feature it is) and a **value** (this says what property the word in question actually has).
- Features therefore come in the form: [ATTRIBUTE: VALUE].
- These features can say lots of things about a given lexical item, e.g. it can say that it has the inherent property of being plural. So *dogs* could have a feature [NUMBER: PLURAL], whereas *dog* would have [NUMBER: SINGULAR].
- The feature will be concerned with right now is the category membership of a particular lexical item. This feature will look like this:

  (6) [CAT(EGORY): {V, N, P, A, …}

- We can think of the lexicon like an unordered collection of words and their properties (just focusing on category for now):

Lexicon

something [CAT: N]		*happy* [CAT: A]
	he [CAT: N]	
danced [CAT: V]		*bought* [CAT: V]

- This is equivalent to the list of phrase structure rules we had that assigned categories to words (e.g. Vt → *bought*, A → *happy*).
- On the Merge-based view of syntax, we can take any two items from the lexicon and put them together with Merge.

- This doesn't mean that every possible combination will result in a grammatical structure, though.
- We still have to constraint the possible combinations to get the trees we want, and rule out the trees we don't want.
- In particular, we have to return to the problem we originally had with our phrase structure rules, namely how do we make sure that a verb like *bought* shows up with a subject *and* an object, and not just a subject. Relatedly, how do we make sure that this doesn't happen for other verbs like *dance*.
- The idea here is that there is a second kind of feature that is associated with lexical items. In addition to a feature that states what their own category membership is, there is another kind of feature which states the category of thing that they need to be combined with to yield a grammatical structure.
- We will call these **selectional features**.
- In order to distinguish selection features from the features that simply state what the category of a given item is, let's make a notational difference to indicate this.
- I will do this by adding •s around the feature.[1]

(7) [•CATEGORY: N•]

- These features reside with the non-selectional version of the category feature on a lexical item.
- A transitive verb like *bought* needs to be combined with a subject and an object, so we need to give it two selectional features. A verb like *dance* only needs a subject, so it gets one selectional feature.
- The other lexical items listed here don't have selectional requirements.
- Our revised lexicon now looks like this:

Lexicon

something [CAT: N]		*happy* [CAT: A]
	he [CAT: N]	
danced [CAT: V, •CAT: N•]		*bought* [CAT: V, •CAT: N•, •CAT: N•]

- So, how are selectional features dealt with? Well, the basic idea is that they need to be **checked** against a matching (non-selectional) feature.

[1] You will see other notations used in other textbooks. Adger uses the prefix *u* in his textbook. So, a feature like the one in (7) would be [*u*N] for him.

- The concept of checking is supposed to express some sense in which these features impose a requirement on the lexical item they reside on. Here, this is the determination of what they have to be combined with in the syntax.
- We will define the condition for checking as being under sisterhood (the relation created by Merge):

Feature checking

A feature [•F: G•] is checked under sisterhood with a corresponding feature [F: G]. Once a feature is checked, it is deleted (under projection).

- The necessity of feature-checking is captured by the following:

Well-formedness condition on trees

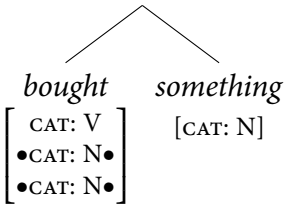
Every feature [•F: G•] must be checked at some point in the tree.

- You can think of [•F: G•] like a 'time bomb' that needs to be defused during the structure building process. If they manage to survive unchecked until the end of the derivation, then
- So, how does this it all fit together? Let's go through the derivation of the following sentence step-by-step:

(8) He bought something.

- In the first step, we pick two items from the lexicon that we want to merge with each other:

(9)



- What is the result of combining these two items? Well, before we had stipulated what the resulting category is (e.g. V + N = V). The Merge-based approach to syntax offers a more general way of capturing this, namely the concept of **projection**.

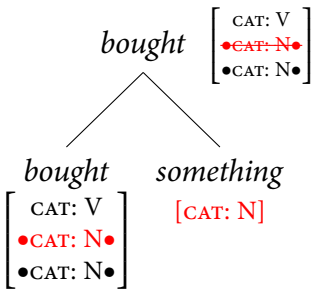
Projection

The node that selects is the one that projects.

- The combination of two lexical items will be treated as inheriting the properties of one of those items. The factor that determines this is selection.
- Let us assume that we take the lexical items *bought* and *something* and put them together with Merge.

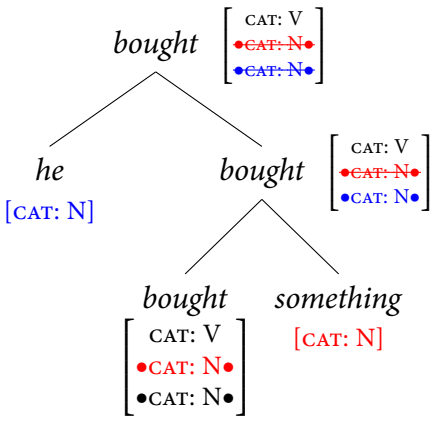
- One of the unchecked selectional features on *bought* matches the categorial feature of its sister *something* (marked in red).
- Since the context for feature checking is given, *bought* will project. Its unchecked features also project, including its own category feature and its remaining selectional feature.
- (I will still mark the checked/deleted selectional feature to keep things clear, but this could, strictly speaking, be omitted.)

(10)



- We can take the structure in (10) and apply another instance of Merge. Taking *he* from the lexicon and merging it with the structure we just built.
- Again, whether or not this newly created constituent has the properties of *he* or *bought something* depends on what selects.
- The remaining selection feature of *bought* is now in a sister relation to an item with a matching feature [CAT: N] (marked in blue).
- Since *bought* bears the selectional feature, it projects again. The other selectional feature is checked (and deleted) under projection.

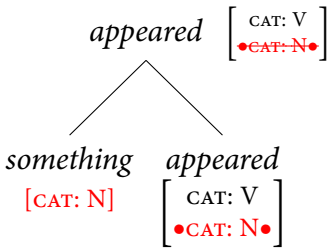
(11)



- So now we have derived a structure for (8). This is not the final structure we want yet, but it already has some advantages. Given the *well-formedness condition on trees* above, if we had not merged a second item with the projection of *bought*, then the final structure would be (10) and ruled out due to the presence of an unchecked feature. This case of under-merging is ruled out.

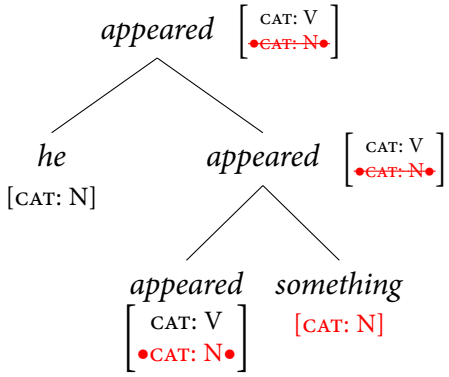
- Similarly, we want to rule out over-merging. Imagine we take the verb *appeared* the lexicon and merge it with *something* (Let's not worry about word order in the tree – we will come back to this):

(12)



- This result is possible since it results in feature checking. But what stops is taking something else from the lexicon and merging it, as below?

(13)



- This has to be ruled out, so we have to assume the Merge is only possible if it results in a feature checking configuration. We add this as condition (a) to our revised definition of Merge below.
- We also include the projection in clause (b) and the pronunciation condition for displacement in (c).

Merge

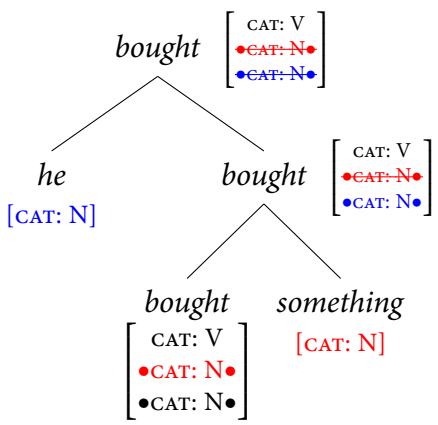
Create a node Z with (immediate) daughters X and Y.

Conditions:
a. Only possible if either X or Y has a selection feature ([•CAT: X•] or [•CAT: Y•])
b. If X selects Y, then Z = X (and *vice versa*)
c. If X is a daughter of Y, then do not pronounce X in Y.

- Now, we have a revised definition of Merge that is very general. Unlike our theory with phrase structure rules, we no longer have to list what every possible combination of X and Y is. Instead, we have introduced a more general idea of projection based on selection.
- This has the advantage that we can add new categories to our grammar and not have to worry about what to call label the constituents we create with them. If they select their sister, then they project. Otherwise, they don't.

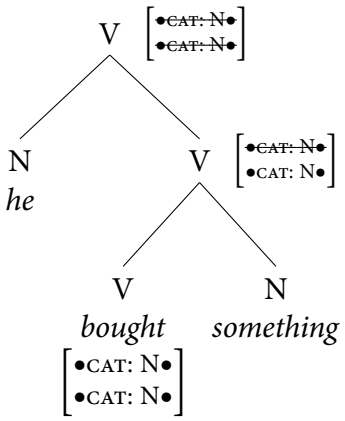
- This approach to phrase structure does not treat trees as the basic primitives of syntactic theory onto which words can be added (like PSRs did), rather it treats every phrase as the projection of a lexical item (≈word).
- The notation we used so far accurately reflects this, but practically the trees will become quite hard to dichper as they get more complex.
- It is common practice to still refer to phrasal constituents (e.g. VP, NP) even on a Merge-based analysis like the current one.
- So let's go back to our tree for *He bought something*:

(14)



- We have said that every lexical item (and its projections) bears a category feature [CAT:X]. Instead of listing this inherent category feature on every projection together with the selectional features, we simply use it as a label for the lexical item in question, as in the tree below.

(15)



- This is very similar to what we had before (but bear in mind we are now using it as a kind of abbreviation for a category feature).
- Now, there are also important distinctions to make. We still want to treat this entire constituent we have built as a *phrase*. The labeling convention we had before, i.e. VP, still applies. Now with features, we have a

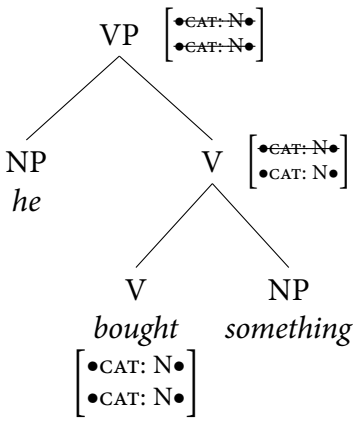
way of defining what a phrase is:

Phrase

A phrase is a node in the tree with no unchecked features.

- We refer to this as the *maximal projection* of a node, it is when the phrase has stopped selecting (and therefore projecting).
- This definition has a useful ambiguity. It will apply both to cases in which a projection of a lexical item had some selectional features but they are now all checked (as in the root node of the tree in (15)), and also to lexical items that did not have selectional features to begin with (such as the selected items *he* and *something*).
- With this in mind, we can update the notation in our tree:

(16)



- Note that we can longer analyze words like *he* with the following structure:

(17)

- This structure cannot be built by Merge! (Neither can a node with three daughters – we will come back to this next week.)
- We can make a number of further distinctions. The lowest projection of a lexical item (assuming it projects) is referred to as the **head** of the phrase. We can also define this based on its feature status.

Head

A head is a node in the tree with no checked features.

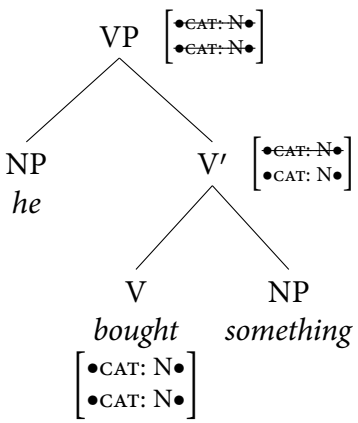
- We will not adapt the notation for this (though sometimes X° is used to indicate the minimal projection of a head).
- The question is what the status of the second projection of V in (17) is. It is neither a head nor a phrase based on our definitions above.
- It has the status of being between a head and a phrase. In order for it to not count as a phrase, it has to have at least one unchecked feature still remaining (this implies it must also have an unchecked feature, too).
- We will call this the **intermediate projection** of a head:

Intermediate projection

An intermediate projection is a node in the tree with at least one checked and one unchecked feature.

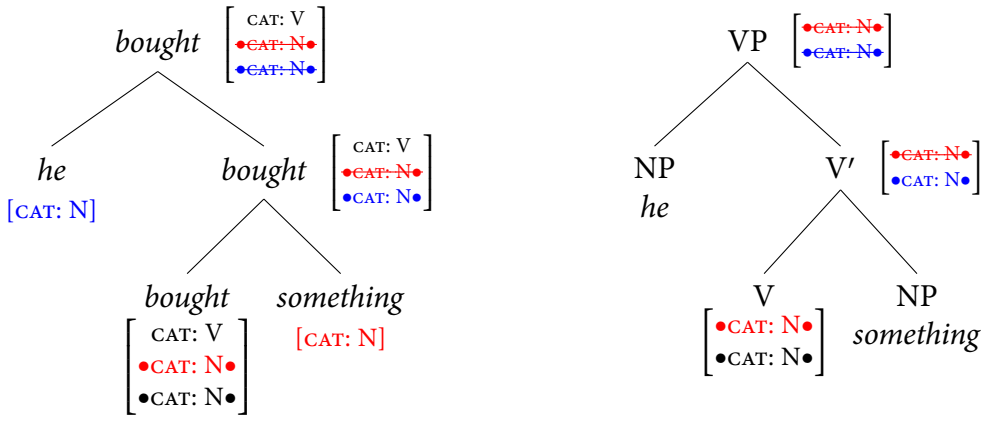
- The notation for this is X′ (read: 'X bar'), where X is the relevant category. (The reason for this is historical: it used to be notated as $\overline{\text{X}}$ where you actually see a 'bar' over the X, but nowadays the aforementioned notation is more common.)
- With this in mind, let's adapt our tree to mark the intermediate projection level V′:

(18)



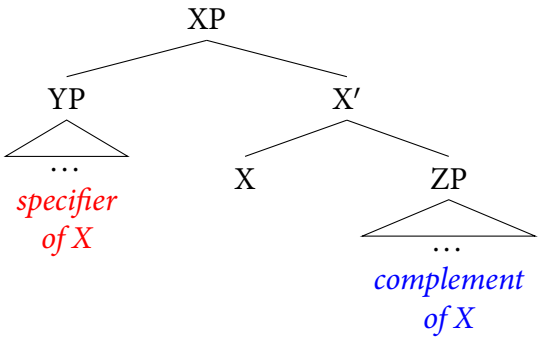
- Now we have revised our assumptions about phrase structure to make three distinctions that we can summarize below:
- Bear in mind the trees below are equivalent ways of representing the result of a verb selecting its arguments:

(19)



- This is the basic of phrase structure we will work with going forward. We have moved away from the idea that phrases are constructed via a set of rules stating what daughters a particular node can have, and we will instead view syntactic constituents as projections of some lexical item.
- The notation we adopted above comes from *X-bar Theory* proposed by Noam Chomsky in the 70s.
- Since we now treat all phrases as projections of heads, we can define an additional useful relation between a head and the phrases it selects.
- We call the first phrase that a head merges with the **complement** of that head. Phrases that are sisters to heads are always complements.
- Any phrases that are subsequently selected by that head are a **specifier** of that head. Specifiers are always sisters to intermediate projections.

(20)



- We have introduced quite a bit of new terminology this week that we can summarize in the table below:

(21)

Node type	Projection type	Label	Description
head	minimal	X (/X°)	contains no checked features
	intermediate	X′ (/$\overline{\text{X}}$)	contains at least one checked/unchecked feature
phrase	maximal	XP	contains no unchecked features

Complement

The complement of a head X is the sister of X

Specifier

The specifier of X is a sister to X′

3 C-command

- The final thing I'd like to discuss is another tree-geometric relation that will be important for us going forward.
- Recall we have previous introduced the concepts of *domination* and *sisterhood*.
- (A given node dominates all of the nodes to which there is a strictly downward path from it. Two nodes are sisters if they share a common (immediate) mother.)
- The new relation we will see is a combination of these two, namely **c-command** (the stands for 'constituent', but this is not relevant for us):

C-command

X c-commands Y if…

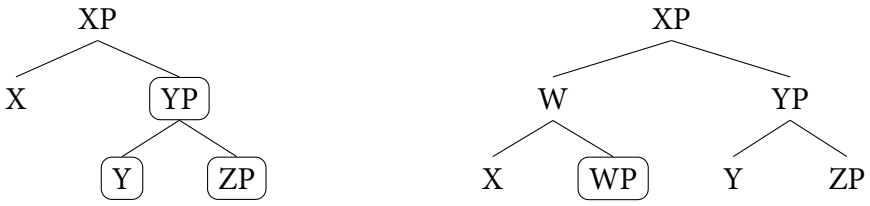
a. Y is the sister of X

or

b. Y is dominated by X

- This rule gives us a different kind of relation between nodes in a tree. In each of the trees in (22), X c-commands all the boxed nodes:

(22)



- We will see that this particular relation seems to matter for several empirical phenomena.
- Consider the following sentences:

(23) a. Steve embarrassed himself at the party.
b. *You embarrassed himself at the party.
c. *Himself embarrassed Steve at the party.

- In (23a), we notice that the word *himself* is restricted in a way that *him* is not. *Himself* must refer to Steve here, it cannot refer to anyone else. This is unlike when we use *him*, which could refer to some other person. (Interestingly, *him* cannot refer to Steve here! We will put that aside for now.)
- The sentence in (23b) is unacceptable. Replacing it with *him* makes the sentence sound natural again. So, it seems like *himself* has a particular restriction that it must be able to refer to something within the same sentence. (We can think of 'being able to refer' to mean something like matching the pronoun, assuming that *himself* restricts reference to male-identifying individuals that are neither the speaker nor addressee in the conversation.)
- Furthermore, (23c) shows that it is not enough that there is a matching individual (referent) mentioned in the same sentence, *himself* may not linearly precede the mention of that referent.
- On top of this, consider the sentence in (24):

  (24) Steve made Jake embarrass himself at the party.

- Here, there are two possible referents that *himself* could pick up, but it is only possible for *himself* to refer to *Jake*.
- This suggests that it is not enough that *himself* is linearly preceded by a matching referent in the same sentence, it must also be the **closest** matching referent in that sentence.
- This view is further supported by the sentence below where we have switched the positions of *Jake* and *himself*:

  (25) Steve made himself embarrass Jake at the party.

- When we do this, we see that *himself* must now refer to Steve.
- What we have said about *himself* is also true of the entire series of pronouns ending in *-self*, e.g. *myself, yourself, herself, itself, oneself, themselves*. Let's try to capture the restriction on what it can refer to with the following rule:

> ***self*-rule**
>
> A pronoun containing *self* must refer to the closest linearly-preceding matching referent.

- We should be a little suspicious about such a rule though (especially in a Syntax class!). We have already seen with the rule to form questions that things that appear to rely on linear order may actually require reference to syntactic structure.
- We will argue that this is also such a case. To see this, consider now the following sentence:

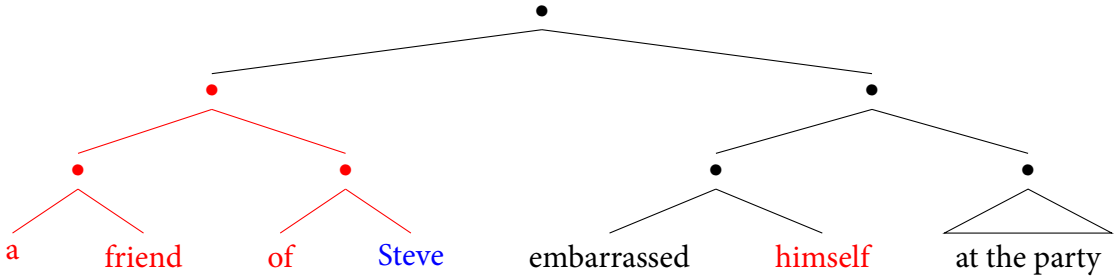
  (26) {A friend of Steve / A friend of Steve} embarrassed himself at the party.

- Why can't *himself* refer to *Steve* in such cases? It looks like the closest preceding antecedent (it is two words closer than *friend*, for example, if we think about things linearly).
- The answer is because what matters here is actually not linear precedence, but c-command.
- This means we have to revise our *self*-rule accordingly:

> ***self*-rule**
>
> A pronoun containing *self* must refer to the closest c-commanding matching referent.

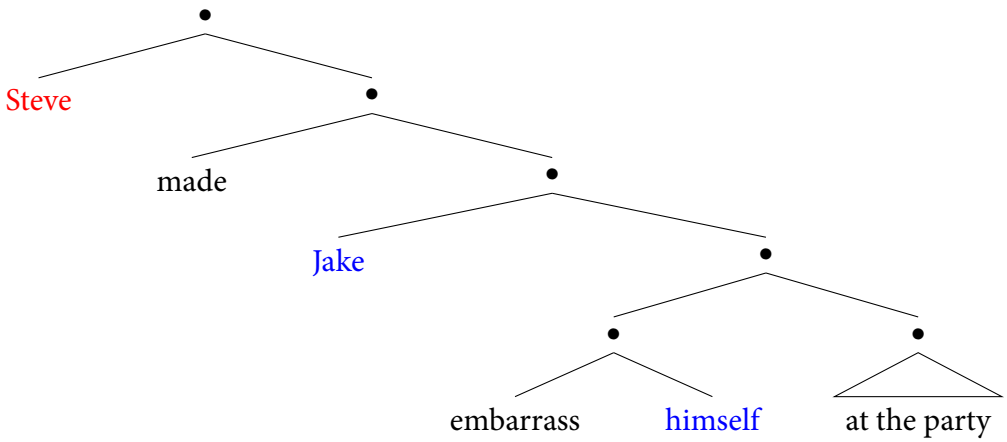
- Now we can see how this captures the fact that *himself* cannot refer to *Steve* in (26). There is no c-command relation between the two nodes. Here, the closest c-commanding referent is the entire phrase *A friend of Steve*.

(27)



- In the case of (25), both *Steve* and *Jake* c-command *himself*, but *Jake* is the closest (measure in terms of the shortest path through the tree).

(28)



- This is just one case in which it seems necessary to refer to c-command.
- Another that I will not discuss in detail here is the possibility of the word *any*.
- I will elaborate on this in next week's class, but here is the crucial data if you want to think about it:

(29) a. Nobody won all of the prizes
b. Nobody won any of the prizes
c. *Anybody won all of the prizes
d. *Anybody won none of the prizes

(30) a. Nobody that the principal likes won all the prizes
b. Nobody that the principal likes won any of the prizes
c. The principal that nobody likes won all of the prizes
d. *The principal that nobody likes won any of the prizes