CAS LX 522 Syntax I

Agree, head movement, and the strength of features (5.4)

11

Inflecting verbs

- Returning now to the question of how the verb comes to look the way it does.

1) Pat ate lunch.

2) Pat eats lunch.

3) Pat has eaten lunch.

4) Pat was eating lunch.

5) Pat might have been eating lunch.

Affix hopping

- Each auxiliary seems to control the form of the form that follows it. We can include T in this generalization as well.

Pat	(T)	eat
	s	

Pat	(T)	have	eat
	s	en	

Pat	(T)	is	eat
	s	ing	

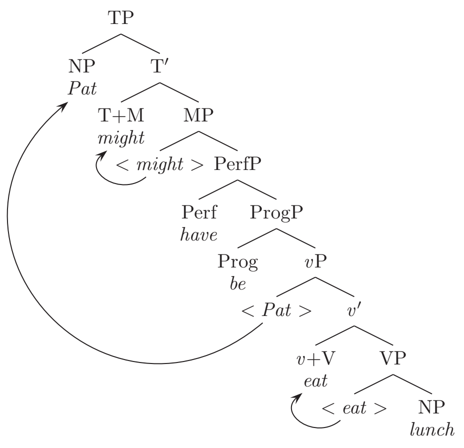
Pat	(T)	have	be	eat
	s	en	ing	

might have been eating

Now, look at how these appear in the tree.

Basically, certain things (T, M, Perf, Prog) assign a verbal form to the next thing (M, Perf, Prog, v) down.

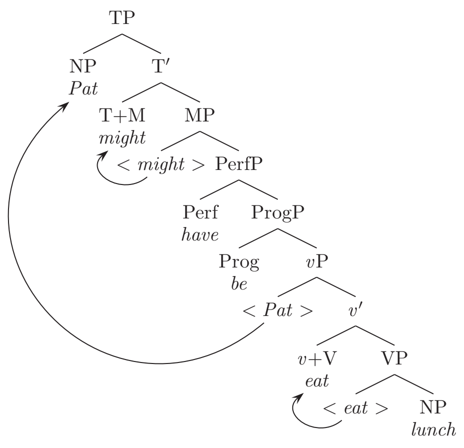
This is a *little* bit like the assignment of reference through binding.



might have been eating

The way we'll model this is by supposing that certain forms take endings. Inflectional endings. Like *en*, *ing*, *s*, etc.

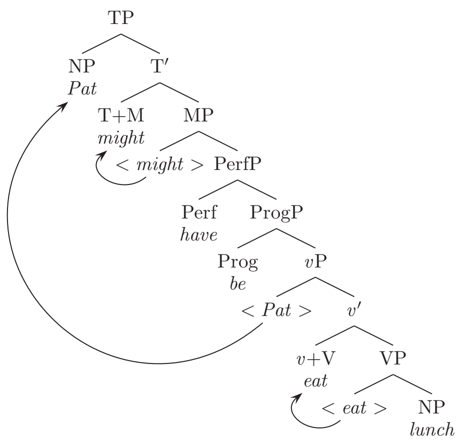
Specifically, suppose that the inflectional ending is represented by an *inflectional* feature, like [Infl: Perf], or [Infl: Prog], or [Infl: Past].



might have been eating

The form comes out of the lexicon without a specific ending, though—what ending it gets is determined *after* it is Merged into the tree, by the next thing up.

That is: whether *eat* comes out as *eats* or *eaten* or *eating* depends on whether the next thing Merged is T, Perf, or Prog.



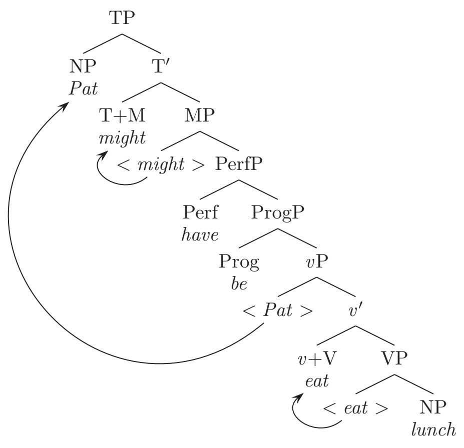
might have been eating

So, at the point where, say, Prog is first Merged into the structure, its Inflectional feature is *unvalued*.

It will be valued by the next thing Merged.

We will also assume that an unvalued inflectional feature is uninterpretable. It must be fixed.

[*u*Infl:]



Agree & unvalued features

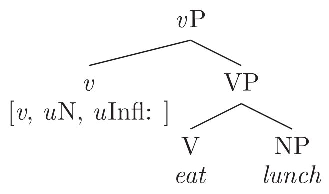
- The idea is that a lexical item might have an *unvalued* feature, which is uninterpretable as it stands and needs to be given a *value* in order to be interpretable.

Agree
In the configuration
X[F: val] … Y[*u*F:]
F **checks** and **values** *u*F, resulting in
X[F: val] … Y[~~*u*~~F: val].

- This gives us **two kinds of uninterpretable features** (unvalued and regular-old uninterpretable features), and two ways to check them (valuing for unvalued features, checking under sisterhood for the other kind).
    - Unvalued [*u*F:]. Regular-old [*u*F].

eat_?

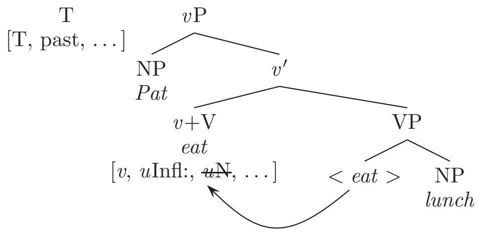
So, *v* has a [*u*Infl:] feature.



past + eat_?

If T is Merged next, it will determine the inflection that will go on the verb. If T is [past], then the verb will become *ate*.

So, T *values* the [*u*Infl:] feature of *v*. As [past], or [pres].



ate

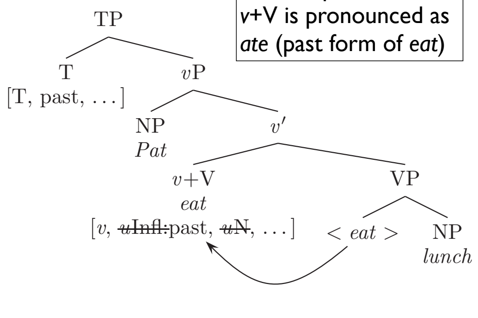
Now, Infl is valued (and is no longer uninterpretable).

Let's suppose that everything that has an inflectional ending of this sort has a [*u*Infl:] feature, then.

That is: Prog, Pres, M, and *v* all have a [*u*Infl:] feature.

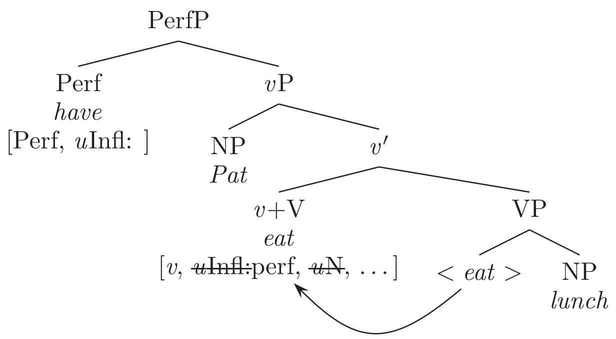
And T, M, Prog, and Pres can value that feature.

Pronunciation:
T is not pronounced,
v+V is pronounced as *ate* (past form of *eat*)



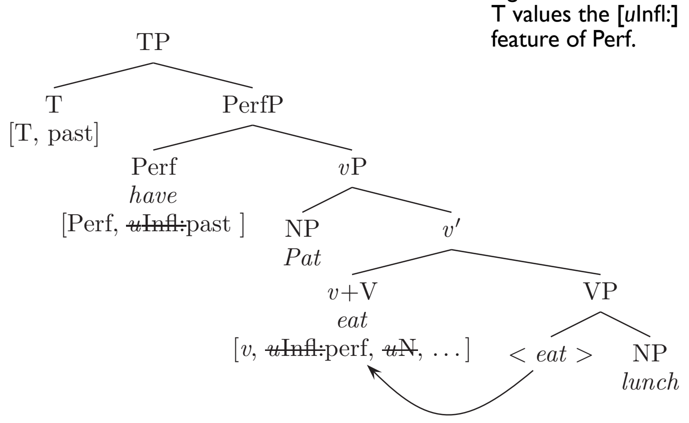
have_? + eaten

Agree:
Perf values the [*u*Infl:] feature of *v*.



had + eaten

Agree:
T values the [*u*Infl:] feature of Perf.



What has [uInfl:], what can value [uInfl:]

- Things of these categories have [*u*Infl:] features:
  - *v*, M, Perf, Prog
- [*u*Infl:] features can be valued (via Agree) by:
  - Tense features (past, present) of T. -*s* or -*ed*.
  - Perf feature of Perf. -*en*.
  - Prog feature of Prog. -*ing*.
  - M feature of M. -Ø (silent)

1) Pat [past] ha-d be-en eat-ing lunch.

The basic operations

- Take some lexical items (a "numeration" or "lexical array")
- Combine any two of them (Merge) to make a new item.
  - Lexical items can have uninterpretable features. Merge can check these features. All of the uninterpretable features must be checked by the end of the derivation.
- Attach one to another (Adjoin).
  - Adjoin does not check features.
- Move stuff around.
  - What can you do? What *can't* you do? Does it check features? Why do you do it? What's really happening?

Move

- There are two basic kinds of movement. We've seen examples of each.
- One is **head-movement**, where a head moves up to join with another head.
  - Examples: V moves to v, {Perf/Prog/M} moves to T
- The other is **XP-movement**, where a maximal projection (an XP) moves up to a specifier of a higher phrase.
  - Example: The subject moving to SpecTP.

Solving a problem via movement

- We will assume that, like with Merge, Move occurs to "solve a problem." And the main problem our system has is unchecked uninterpretable features. So, Move must check features.
- We have two ways to check features so far. One of them is under sisterhood (Merge). The other is "at a distance" (Agree).
- What kind of problem could Move solve? Well, for one thing, it must not be able to solve the problem in place, without moving. Seems to need "closeness."

Two existing means of checking features

- P has a [*u*N] feature. Merge it with an N(P), and the [*u*N] feature of P is checked.

c-selection
If X[F] and Y[*u*F] are sisters, the *u*F feature of Y is checked: Y[~~*u*F~~].

- T has a [tense:past] feature.
- Strictly speaking [tense:past] doesn't look like it's a valued [Infl] feature. We need to *stipulate* in addition a list of things that can value [Infl] features.

inflection
If X[F] c-commands Y[*u*F:] the *u*F feature of Y is valued and checked: Y[~~*u*F~~:val].

Generalizing Agree

- Agree requires:
  - An uninterpretable or unvalued feature
  - A matching feature
  - Line of sight (c-command)
- And results in:
  - Valuing of unvalued features.
  - Checking of the uninterpretable features.

- Our first version of checking (sisterhood) is a special case of this more general conception of Agree.
  - Except that we *do* want the [*u*N] feature of P to be checked by directly Merging P and an NP—not "at a distance" like agreement.

Strong features

- In order to check the [*u*N] feature of P *only* through Merge (sisterhood), we will define a special kind of uninterpretable feature: the **strong** feature.
- A strong feature can only be checked when the matching feature is on an element that shares the same mother node.
- We will write strong features with a *:
  - P [P, *u*N*]
  - C-selection features are strong.

Generalizing Agree

- Matching:
  - Identical features match. [N] matches [*u*N].
  - Some features match several things. [*u*Infl:] can match values of the [tense] feature ([tense:pres], [tense:past]), as well as the category features [Perf], [Prog], [M].
  - What if there are two options? We'll see later that only the closest one participates in Agree.
- Valuing/Checking:
  - An unvalued feature is always uninterpretable.
  - Valuing a feature will check it.
  - A privative feature is simply checked when it matches.

Other properties of Agree (mainly relevant later)

- **Strong features Agree first.**
- Where a single head has more than one feature that must Agree, the strong ones go first.

The system is lazy.

- Agree always goes with the closest option it can find in order to check an uninterpretable feature.
- If Agree locates a matching feature on X for one uninterpretable feature, and X has a different feature that also matches, both features will be checked.
- Examples are coming up later, but for cross-referencing: these properties are important for subject agreement.

Agree

- If:
  - X has feature [F1], Y has feature [F2]
  - X c-commands Y or Y c-commands X
  - [F1] and/or [F2] are/is uninterpretable.
  - [F1] matches [F2]
  - X and Y are close enough, meaning:
    - There is no closer matching feature between X and Y.
    - If [F1] or [F2] is strong, X and Y share the same mother node
- Then:
  - Any unvalued feature ([F1] or [F2]) is valued.
  - The uninterpretable feature(s) is/are checked.

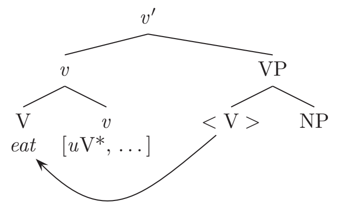
Comments on Agree

- This statement of Agree allows for several different configurations:

[*u*F]…[F]	[F]…[*u*F]	[*u*F]…[*u*F]
c-selection	Inflection	Case

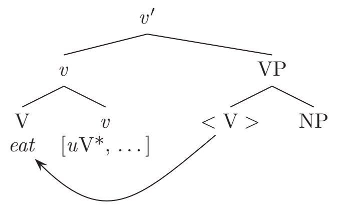
- Strong features must be checked very **locally**.
  - Merge can provide this locality.
  - Move can also provide this locality.
    - **Strong features are what motivates movement.**

V+v=?



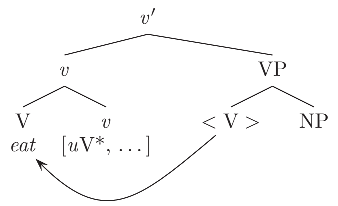
- When V moves to *v*, they combine in a way that we have been writing just as V+*v*. Let's be more precise.
- In fact, we assume that V **head-adjoins** (adjoins, head-to-head) to *v*. This is the same sort of structure that Adjoin creates between maximal projections.
  - The *v* head is replaced by the *v* head with V adjoined.
- Adjunction does not change projection levels—*v* is still a minimal projection, still the head of *v*P. But it is a **complex head** (it's a *v* with a V adjoined to it).

V+v=?



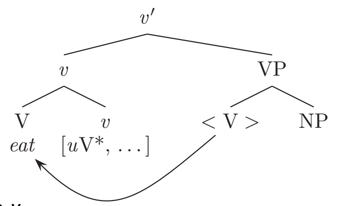
- What happens to the VP from which the V moved?
- It is still a VP, it must still have a head. The features of the VP are the features of the head (recall for example, that checking the uninterpretable feature on the head is the same as checking the uninterpretable feature on the projection of the head). The VP is still a *VP*, its head is still a verb (with category feature [V]), and presumably all the rest of the features as well.
- We notate the original location of the V by writing <V> (standing for the "trace" left behind by the original V). But since <V> must still be a bundle of features, the same one that was there before movement, <V> is really just another copy (or, well, the original) of the verb.

V+v=?



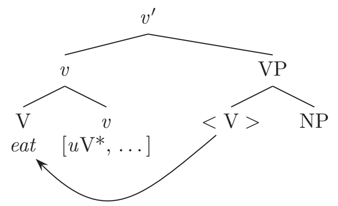
- **Moral: "Head-movement" can be viewed as Copy+Adjoin.**
- Make a copy of V. Replace the original *v* is replaced by the syntactic object formed by Adjoining the copy of V to *v*.
- If *v* has a [*u*V*] feature, this puts V close enough to *v* to check that feature. This is *why* we move V.
- Note: This appears to make a change *inside* the object. Merge always happens at the root. However: Think about the root. It has the features of *v*, its head. It is a projection of *v*. There is a sense in which this is still affecting only the root node, it's adjunction to its head.

V+v=?



- We always move V to *v*.
- Reason:
  v always has a [uV*] feature.
- But why wasn't this checked when we Merged *v* and VP? (Like the [*u*N*] feature of P is checked when we Merge P and NP…)
- The Hierarchy of Projections says that *v* > VP: When you finish VP, you Merge it with *v*. Only then do you Move and Merge with other things. The HoP takes priority.
- When you Merge two nodes in order to satisfy the HoP, you don't get to Agree. You have to move to the next step (Merge or Move).

V+v=?



- **That's craziness, isn't it? Now instead of one V, we have two identical copies. Why don't we get Pat Pat ate ate lunch?**
- We need both copies (the higher one to check the feature, the lower one to head the original projection of V). But on the other hand, the verb was picked from the lexicon just once.

  A-P interface: Only the highest copy is pronounced.
- Spelling out the idea that you "move it but leave a trace." Highest copy = the one that is not c-commanded by another copy. A head V adjoined to another head *v* c-commands the same nodes that *v* did. This is a stipulation, but if we define c-command in a more complicated way, it comes to this. A general property of adjuncts is that they are "just as high" in the tree as the thing they adjoined to, so they "see" (c-command) the same stuff as the thing they adjoined to.

A note on node labeling

- A node is labeled as a maximal projection (XP) if there are no more *strong* features left to check.
  - Notice that *v* has [*u*Infl:] even when we're finished with it and Merge it with the next head up (M, Perf, Prog, Neg, or T). But we still want there to be a *v*P.
  - C-selection features (like the [*u*N*] feature(s) of V, or the [*u*N*] feature of P) are always strong.

T has [uN*] ("EPP")

- V moves to *v*:
  - *v* has a [*u*V*] feature (always).
- Moving the subject from Spec*v*P to SpecTP:
  - T has a [*u*N*] feature (always).
  - Moving the subject (making a copy and Merging it with T) put the N feature of the subject close enough to T for the [*u*N*] feature to be checked.

As for why you don't satisfy the [*u*V*] feature of *v* the same way, by moving VP into Spec*v*P, we could speculate, but there's no particularly satisfying answer. We'll set that aside.