# CAS LX 422 / GRS LX 722 Intermediate Syntax

8

TP, Agree, and our quickly growing tree
(5.1-5.3)

## Auxiliaries and modals and verbs

I ate.

I could eat.

I had eaten.

I was eating.

I had been eating.

I could have eaten.

I could be eating.

I could have been eating.

So: *could*, *have*, *be*, *eat*.
How do we determine what form each verb takes?

## Auxiliaries and modals and verbs

*Have*: Perfective (aspect)

I have eaten. I had eaten.

*Be*: Progressive (aspect)

I am eating. I was eating.

*Could*: Modal

- I can eat. I could eat. I shall eat. I should eat. I may eat. I might eat. I will eat. I would eat.

## Auxiliaries and modals and verbs

- I could have been eating.
- *I could be having eaten.
- *I was canning have eaten.
- *I had cannen be eating.
- *I was having cannen eat.
- *I had been canning eat.

It looks like there's an order:

- Modal, Perf, Prog, verb.

## Auxiliaries and modals and verbs

Suppose:

*Have* is of category Perf.

*Be* is of category Prog.

*May*, *might*, *can*, *could* are of category M.

They are heads from the lexicon, we will Merge them into the tree above *v*P. Their order is captured by a new extended Hierarchy of Projections:

- Modal > Perf > Prog > *v* > V

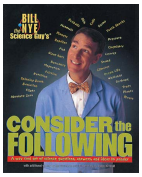
Except not every sentence has these. So:

- (Modal) > (Perf) > (Prog) > *v* > V

## Negation

Consider the following:

- I did not eat.
- I could not eat.
- I had not eaten.
- I was not eating.
- I had not been eating.
- I could not have been eating.




Suppose *not* is of category Neg.

How do we describe where *not* occurs? How can we fit it into our Hierarchy of Projections?

# Where does Neg fit?

Suppose that we *can* fit Neg in our Hierarchy of Projections. Just like the other things we just added.

- (Modal) > (Perf) > (Prog) > v > V

Where would it go in the HoP, and how can we explain the word order patterns?

- I could not have been eating.
- I had not been eating.
- I was not eating.
- I did not eat.

Remember *v* and how we explained where the verb is in *Pat gave a book to Chris*?

# A-ha.

Picture this:

- I ?+might not <might> have been eating.
- I ?+had not <had> been eating.
- I ?+was not <was> eating.

So what is ?, then?

- He did not eat. He ate.
- He does not eat. He eats.

All that *do* seems to be doing there is providing an indication of…tense.

# HoP revisited

So, now we know where Neg goes. Above all the other things, but below tense (category T).

- T > (Neg) > (M) > (Perf) > (Prog) > *v* > V

Just as V moves to *v*, so do Perf, Prog, and M move to T.

If Neg is there, you can see it happen.

- They T+shall not <shall> be eating lunch.
- They T+shall <shall> be eating lunch.

# What does do do?

But what about when there's just a verb and Neg, but no M, Perf, or Prog?

- I ate lunch.
- I did not eat lunch.

*Eat* clearly does not move to T.

But *not* "gets in the way", so tense cannot "see" the verb. Instead, the meaningless verb *do* is pronounced, to "support" tense. "*Do*-support"

- We will return to the details in due course…

# So, we have T

We've just added a category T, tense.

- **The idea:** The tense of a clause (past, present) is the information that T brings to the structure.

  T has features like [T, past] or [T, pres]

  Or perhaps [T, past] or [T, nonpast].

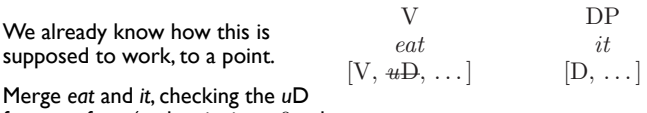
  These features are *interpretable* on T. T is where tense "lives." We see reflections of these tense features on verbs (*give*, gave, *go*, *went*) but they are just reflections. Agreement. The interpretable tense features don't live on verbs, they live on T.

# They might eat it.

*they* [D, …] *v* [*v*, *u*D, …]
*eat* [V, *u*D, …] *it* [D, …]
workbench
*might* [M, …] T [T, past]

We already know how this is supposed to work, to a point.

Merge *eat* and *it*, checking the *u*D feature of *eat* (and assigning a θ-role to *it*, namely Theme—this is DP daughter of VP).



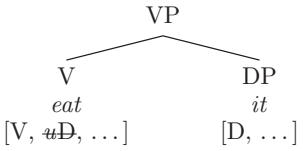
# They might eat it.

*they* [D, …]  *v* [*v*, *u*D, …]

workbench

*might* [M, …]  T [T, past]

VP [V, ...]




We already know how this is supposed to work, to a point.

Merge *eat* and *it*, checking the *u*D feature of *eat* (and assigning a θ-role to *it*, namely Theme—this is DP daughter of VP).

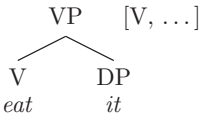
# They might eat it.

*they* [D, …]  *v* [*v*, *u*D, …]

VP [V, ...]

workbench

*might* [M, …]  T [T, past]




We already know how this is supposed to work, to a point.

Merge *eat* and *it*, checking the *u*D feature of *eat* (and assigning a θ-role to *it*, namely Theme—this is DP daughter of VP).

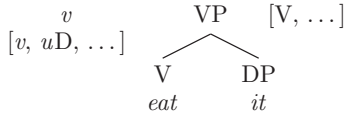
# They might eat it.

*they* [D, …]  *v* [*v*, *u*D, …]

VP [V, ...]

workbench

*might* [M, …]  T [T, past]




Merge *v* and the VP *eat it*, in conformance with the Hierarchy of Projections. *v* projects, and still has a *u*D feature.

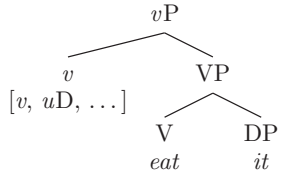
# They might eat it.

*they* [D, …]

workbench

*might* [M, …]  T [T, past]

*v*P [*v*, *u*D, …]




Merge *v* and the VP *eat it*, in conformance with the Hierarchy of Projections. *v* projects, and still has a *u*D feature.

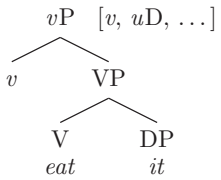
# They might eat it.

*they* [D, …]

*v*P [*v*, *u*D, …]

workbench

*might* [M, …]  T [T, past]




Merge *v* and the VP *eat it*, in conformance with the Hierarchy of Projections. *v* projects, and still has a *u*D feature.

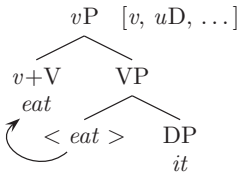
# They might eat it.

*they* [D, …]

*v*P [*v*, *u*D, …]

workbench

*might* [M, …]  T [T, past]

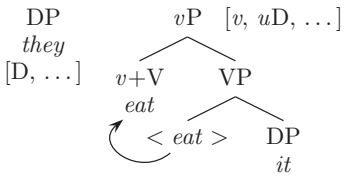



Move the V *eat* up to *v*.

# They might eat it.

*they* [D, …]
*v*P [*v*, *u*D, …]

workbench
*might* [M, …] T [T, past]

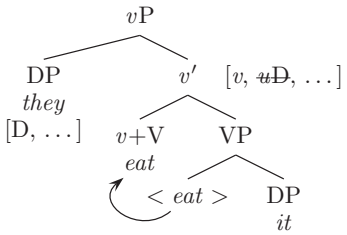



Merge *they* with *v*′ to check the *u*D feature and assign a θ-role (Agent, this is DP daughter of *v*P).

# They might eat it.

workbench
*might* [M, …] T [T, past]

*v*P [*v*, …]

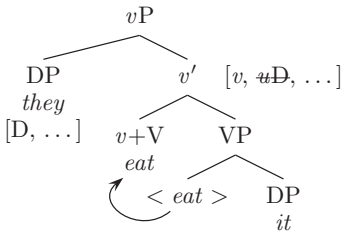



Merge *they* with *v*′ to check the *u*D feature and assign a θ-role (Agent, this is DP daughter of *v*P).

# They might eat it.

*v*P [*v*, …]

workbench
*might* [M, …] T [T, past]




So, now what do we do with *might*?

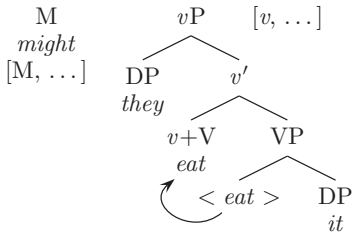
1) And eat it they shall.
2) What they should do is eat it.

It kind of seems like it goes between the subject and the verb, but how?

# They might eat it.

*v*P [*v*, …]

workbench
*might* [M, …] T [T, past]



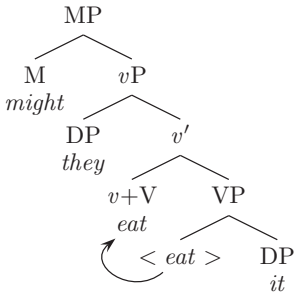

If we leave everything as it is so far (UTAH, Hierarchy of Projections), the only option is to Merge *might* with the *v*P we just built.

So, let's.

# They might eat it.

workbench
T [T, past]

MP [M, …]



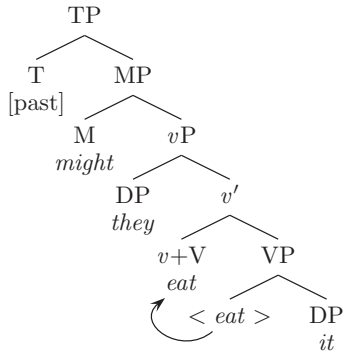

- Now, we have one more thing on our workbench (T) and the HoP says that once we finish with M, we Merge it with T.
- And so Merge T, we shall.

# They might eat it.

Then, M moves up to T.

Why? Because M, Perf, and Prog all move up to T. For the same kind of reason that V moves up to *v*.

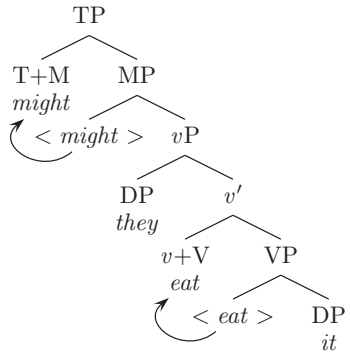
Right now we have no way to describe this in our system, except with this "rule from the outside" that stipulates that V moves to *v*, and {M/Perf/Prog} moves to T.



# They might eat it.

Ok, that's all fine and good, except that the sentence is *They might eat it* not *Might they eat it*

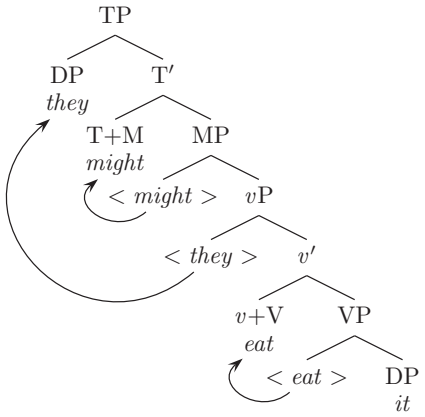
How do we get *They might eat it* out of this?




# They might eat it.

As previewed earlier, the subject *moves* to this first position in the sentence, around the modal.

"Moving" *they* here means Merging a copy…

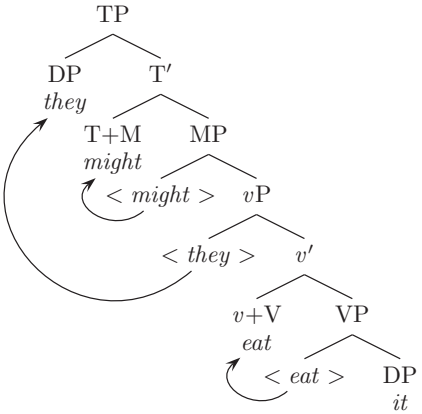



# They might eat it.

Great. Why?

Jumping ahead, we're going to say that this is a property of T-type things generally: T needs to have a DP in its specifier.

We can encode this as a (special type of) uninterpretable feature on T: [*u*D*]. More on that shortly.




# They ate it

Now that we have T in the Hierarchy of Projections, we're stuck with it.

Yet, where is T in *They ate it* or *They eat it*?

It looks like the tense marking is on the verb, we don't see anything between the subject and the verb where T ought to be.

Now that we have T, this is where tense features *belong*. We take this to be the thing that determines the tense of the sentence, even if we sometimes *see* the marking on the verb.

# They ate it

Since (most) verbs sound different when in the past and in the present tense, we suppose that there is a [past] or [present] feature on the verb.

However, to reiterate: **tense belongs on T.**

The tense features on the verbs are uninterpretable.

# Feature classes

There are **tense** features. Like past, like present. There are **case** features. Like nom, like acc. There are **person** features. Like 1st, like 2nd. There are **gender** features. Like masculine, like feminine.

So, we can think of this as a feature category or feature **type** that has a **value**.

[Gender: masculine] [Person: 1st]

[Tense: past] [Case: nom]

# Agree

T nodes have features of the tense type. Maybe past, maybe present.

Suppose that v has an uninterpretable feature of the tense type, but *unvalued*.

What we're trying to model here is *agreement*.

**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].

# 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.

- The statement of Agree on the previous slide is essentially saying just that, formally.

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].

# 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) | be | eat |
|---|---|---|---|
| | s | ing | |

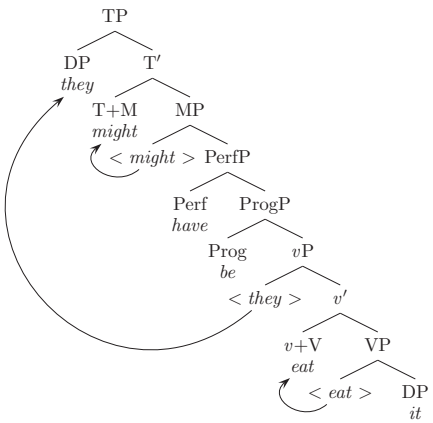
| Pat | (T) | have | be | eat |
|---|---|---|---|---|
| | s | en | ing | |

# 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.

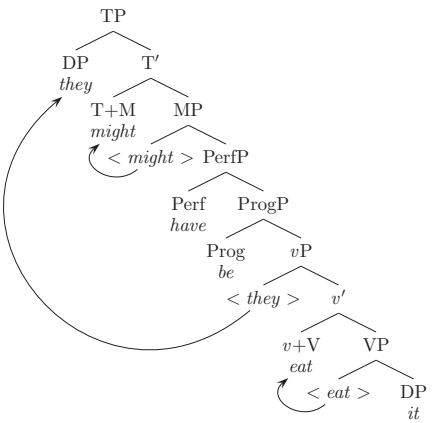
This is a *little* bit like the assignment of reference through binding.




# 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.

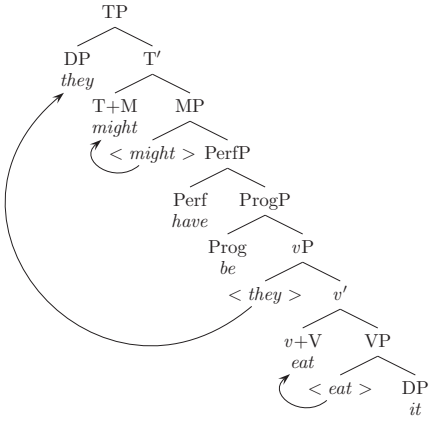
Specifically, suppose that the inflectional ending is represented by an *inflectional* feature, like [Infl: Perf], or [Infl: Prog], or [Infl: Past].



# 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.



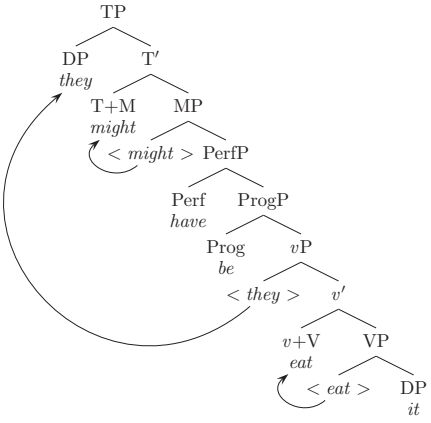

# might have been eating

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.

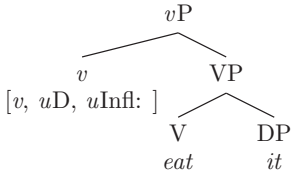
**Agree**
In the configuration
X[F: val] … Y[*u*F: ]
F **checks** and **values** *u*F, resulting in
X[F: val] … Y[~~*u*~~F: val].

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_?

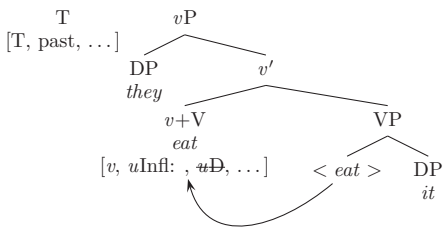
So, *v* has a [*u*Infl: ] feature.




# 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

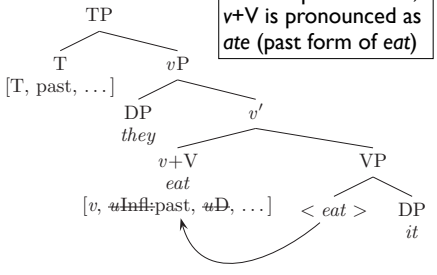
Now, Infl is valued (and is no longer uninterpretable).

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

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

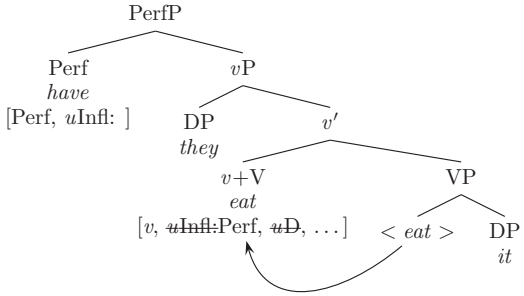
And T, M, Prog, and Perf can value that feature.

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



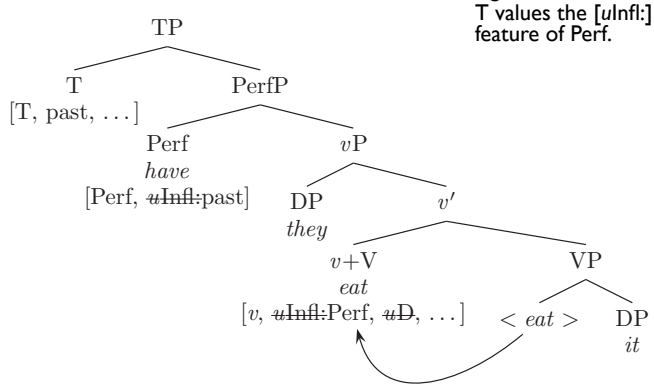
# have_? + eaten

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*D] feature. Merge it with an D(P), and the [*u*D] feature of P is checked.

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.

**c-selection**

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

**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*D] feature of P to be checked by directly Merging P and an DP—not "at a distance" like agreement.

# Strong features

In order to check the [*u*D] 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*D*]

- C-selection features are strong.

# Generalizing Agree

- Matching:
  - Identical features match. [D] matches [*u*D].
  - 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 |

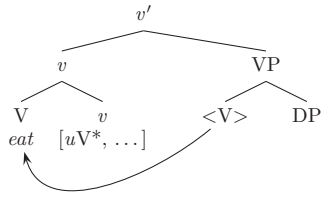
Strong features must be checked very **locally**.

Merge can provide this locality.

Move can also provide this locality.

- **Strong features are what motivates movement.**

# V+v=?



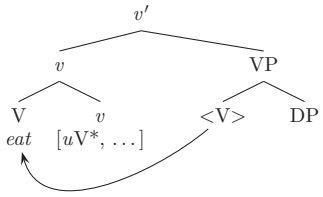

When V moves to *v*, they combine in a way that we have been writing just as V+*v*. Let's be more precise.

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=?

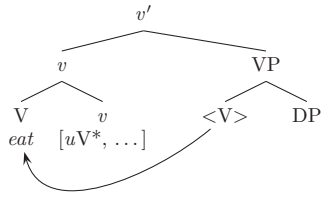



What happens to the VP from which the V moved?

- 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 *V*P, 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=?



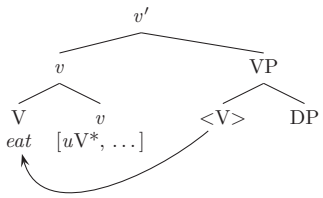

**Moral: "Head-movement" can be viewed as Copy+Adjoin.**

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=?



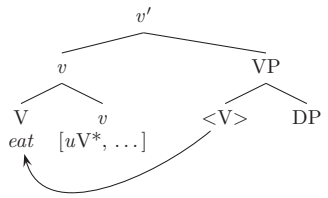

We always move V to *v*.

- Reason:
  **v always has a [uV*] feature.**

But why wasn't this checked when we Merged *v* and VP? (Like the [*u*D*] feature of P is checked when we Merge P and DP…)

- 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*D*] feature(s) of V, or the [*u*D*] feature of P) are always strong.

# T has [uD*] ("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*D*] feature (always).

Moving the subject (making a copy and Merging it with T) put the D feature of the subject close enough to T for the [*u*D*] 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.

# Only auxiliaries move to T

1) I do not eat green eggs and ham.
2) I have not eaten green eggs and ham.
3) I have not been eating green eggs and ham.
4) I would not have been eating green eggs and ham.

There is a set of things that move to T—the auxiliaries (*have*, *be*, modals). Main verbs *do not* move to T. Only the *top* auxiliary moves to T.

Movement is driven by strong features.

# Auxiliaries moving to T

Since auxiliaries and main verbs behave differently, they must be differentiated. Suppose auxiliaries have the feature [Aux] ("the property of being auxiliaries").

Auxiliaries move. Movement is driven by a strong feature. But what strong feature?

[*u*Aux*] on T?

- No. That does not work.

[*u*T*] on Aux?

- No. That would not be promising.

# Auxiliaries moving to T

Auxiliaries have a [*u*Infl:] feature, valued by the next thing up.

The topmost auxiliary has its [*u*Infl:] feature valued by T.

The topmost auxiliary is the only auxiliary that moves to T.

An auxiliary whose [*u*Infl:] feature is valued by T will move to T.

Movement is driven by strong features.

It appears that we need to say this:

If a head has the feature [Aux], and

If that head's [*u*Infl:] feature is valued by T,

Then the feature is **valued as strong**.

The auxiliary must move to T to be checked.

T[tense:pres] … *be*[Aux, *u*Infl:]
T[tense:pres] … *be*[Aux, *u*Infl:pres*]
T[tense:pres]+*be*[Aux, *u*Infl:pres*] … <*be*>

# French vs. English

In English, adverbs cannot come between the verb and the object.

1) *Pat eats often apples.
2) Pat often eats apples.

In French it's the other way around.

3) Jean mange souvent des pommes.
   Jean eats often of.the apples
   'Jean often eats apples.'
4) *Jean souvent mange des pommes.

If we suppose that the basic structures are the same, why might that be?

# French vs. English

Similarly, while only auxiliaries in English show up before negation (*not*)…

- John does **not love** Mary.
- John **has not** eaten apples.

…all verbs seem to show up before negation (*pas*) in French:

- Jean (n')**aime pas** Marie.
  Jean (ne) loves not Marie
  'Jean doesn't love Marie.'
- Jean (n')**a pas** mangé des pommes.
  Jean (ne)has not eaten of.the apples
  'Jean didn't eat apples.'

# V raises to T in French

What it looks like is that both V and auxiliaries raise to T in French.

This is a **parametric difference** between English and French.

A kid's task is to determine whether V moves to T and whether auxiliaries move to T.

| | T values [*u*Infl:] on Aux | T values [*u*Infl:] on *v* |
|---|---|---|
| English | Strong | Weak |
| French | Strong | Strong |

# Swedish

Looking at Swedish, we can see that not only do languages vary on whether they raise main verbs to T, languages also vary on whether they raise auxiliaries to T:

- …om hon **inte köpte** boken
  whether she not bought book-the
  '…whether she didn't buy the book.'
- …om hon **inte har** köpt boken
  whether she not has bought book-the
  '…whether she hasn't bought the book.'

So both parameters can vary.

# Typology of verb/aux raising

Interestingly, there don't seem to be languages that raise main verbs but not auxiliaries.

- This double-binary distinction predicts there would be.
- It overgenerates a smidge.

This is a pattern that we would like to explain someday, another mystery about Aux to file away.

- Sorry, we won't have any satisfying explanation for this gap this semester.

| | T values [*u*Infl:] on Aux | T values [*u*Infl:] on *v* |
|---|---|---|
| English | Strong | Weak |
| French | Strong | Strong |
| Swedish | Weak | Weak |
| **Unattested** | Weak | Strong |