CAS LX 522 *Syntax I* **Pat must not have been sending flowers to Chris**

Suppose that you were faced with the task of generating a structure for *Pat must not have been sending flowers to Chris*.

First, assemble your numeration. What are the elements in this sentence?

T	[T, *u*N*, tense:past]	There is always a T in a sentence. T always (in English) has [*u*N*] (the "EPP feature"). *Must* is past tense (cf. *may*), so we know there is [tense:past].
v	[*v*, *u*Infl:, *u*N*, *u*V*]	There is always a *v* in a sentence. *v* always has [*u*V*] (causes V to raise to *v*). *give* is agentive, so this *v* assigns the Agent θ-role; hence it has a [*u*N*] feature.
Not	[Neg]	*not* is of category Neg.
must	[M, *u*Infl:, Aux]	We have the modal *must*, which is M. M always has a [*u*Infl:] feature, and all modals except the infinitive marker *to* have the [Aux] feature.
Perf	[Perf, *u*Infl:, Aux]	We have perfective *have*, which is Perf. Perf always has [*u*Infl:] and [Aux] features.
Prog	[Prog, *u*Infl:, Aux]	We have progressive *be*, which is Prog. Prog always has a [*u*Infl:] and [Aux] features.
send	[V, *u*N*, *u*P*]	*send* is a ditransitive (that is, between V and *v* there are three θ-roles to give out. *v* is responsible for Agent, and V is responsible for the other two. Hence, we have [*u*P*] corresponding to the Goal θ-role and [*u*N*] corresponding to the Theme θ-role).
to	[P, *u*N*]	We have the preposition *to*. Prepositions always have [*u*N*] for their object.
Chris	[N]	
Pat	[N]	
flowers	[N]	

Then, start at the bottom (generally the right edge of the sentence in an SVO language like English), and work your way up.

Step 1. Merge *to* and *Chris*.

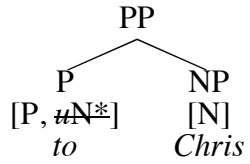
This puts the [*u*N*] feature of *to* in close proximity to the [N] of *Chris*:

[*u*N*] is checked (it was strong, but the locality condition is satisfied).

Since *to* had its feature checked (it motivated the Merge), the features of *to* project.

Since *Chris* does not project further and had no strong features to check, it is a maximal projection (NP).

Since there are no further uninterpretable strong features on *to*, the object we just formed is a maximal projection (PP).



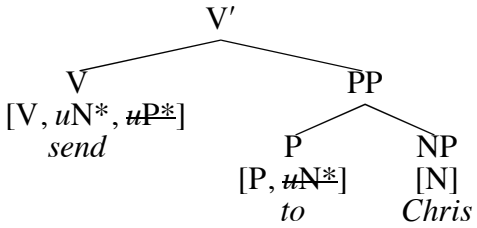
Step 2. Merge *send* and the PP.

This puts the [*u*P*] feature of *send* in close proximity to the [P] of the PP. [*u*P*] is checked.

Since *send* had its feature checked, the features of *give* project.

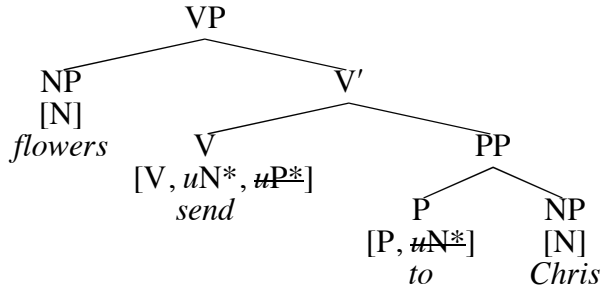
Since there is still a strong uninterpretable feature on *send*, this is an intermediate projection (V′).

The UTAH tells us that *to Chris* is the Goal (PP daughter of V′).



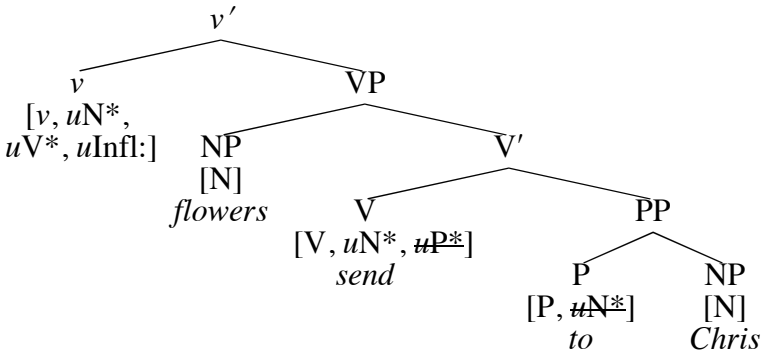
Step 3. Merge *flowers* and the V′.

This puts the remaining [*u*N*] feature of *send* close to the [N] feature of *flowers*. [*u*N*] is checked.
Since *give* had its feature checked, the features of *send* project.
Since there are no more strong uninterpretable features on *send*, this is a maximal projection (VP).
Since *flowers* does not project further and had no strong features to check, it is a maximal projection (NP).
UTAH tells us that *flowers* is the Theme (NP daughter of VP).



Step 4. Merge *v* and VP.

We are finished with V now, and *v* is the next step up on the Hierarchy of Projections.
No features are checked in this step.
Because *v* is higher than V on the hierarchy of projections, the features of *v* project.
Since *v* still has a strong uninterpretable feature to check, this is an intermediate projection (*v*′).



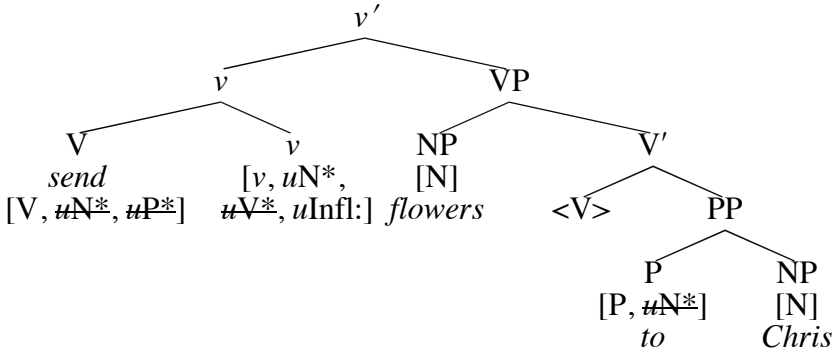
Step 5. Move V to *v*.

v has a strong uninterpretable [*u*V*] feature. It was not checked in Step 4 because it did not motivate that Merge (the Hierarchy of Projections did).

Because the feature [*u*V*] is strong, it can only be checked if it is local to the [V] feature that checks it. Sisters are local. But *send* is not local to *v*.

We move V to *v* by copying V and adjoining it to *v*.

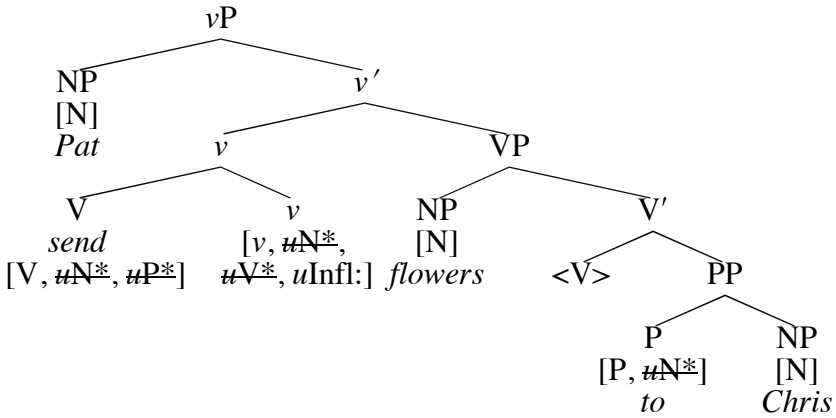
We represent that by putting <V> in the original location of the V, and replacing *v* with the complex head V+*v* as shown below (V adjoined to *v*).



Step 6. Merge *Pat* and *v′*.

This puts the remaining [*u*N*] feature of *v* close to the [N] feature of *Pat*. [*u*N*] is checked.
Since *v* had its feature checked, the features of *v* project.
Since there are no more strong uninterpretable features on *v*, this is a maximal projection (*v*P).
Since *Pat* does not project further and had no strong features to check, it is a maximal projection (NP).
UTAH tells us that *Pat* is the Agent (NP daughter of *v*P).

Sometimes NP daughter of *v*P is actually an Experiencer—if it is, we assume that it is a slightly different *v* ($v_{\text{EXPERIENCER}}$) for which NP daughter of *v*P is an Experiencer. But with *send* here, we have v_{AGENT}, not $v_{\text{EXPERIENCER}}$.

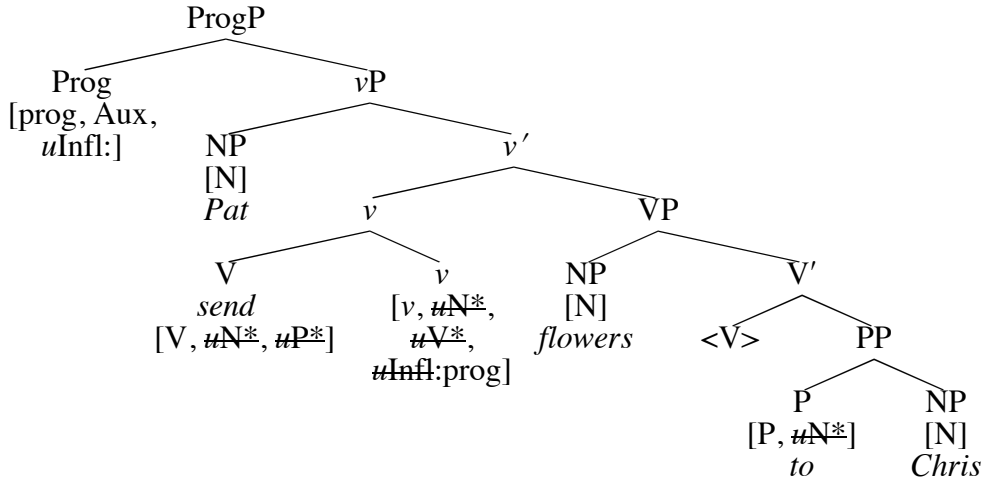


Step 7. Merge Prog and *v*P.

We are finished with *v* now, and Prog is the next step up on the Hierarchy of Projections.
Because Prog is higher than *v* on the hierarchy of projections, features of Prog project.
Since Prog has no strong uninterpretable features to check, this is a maximal projection (ProgP).
The category feature of Prog ([prog]) can value an uninterpretable [*u*Infl:] feature.
Prog now c-commands *v*, which has a [*u*Infl:] feature still unvalued.
[prog] matches and values [*u*Infl:] on *v* resulting in [*u*Infl:prog] on *v*.

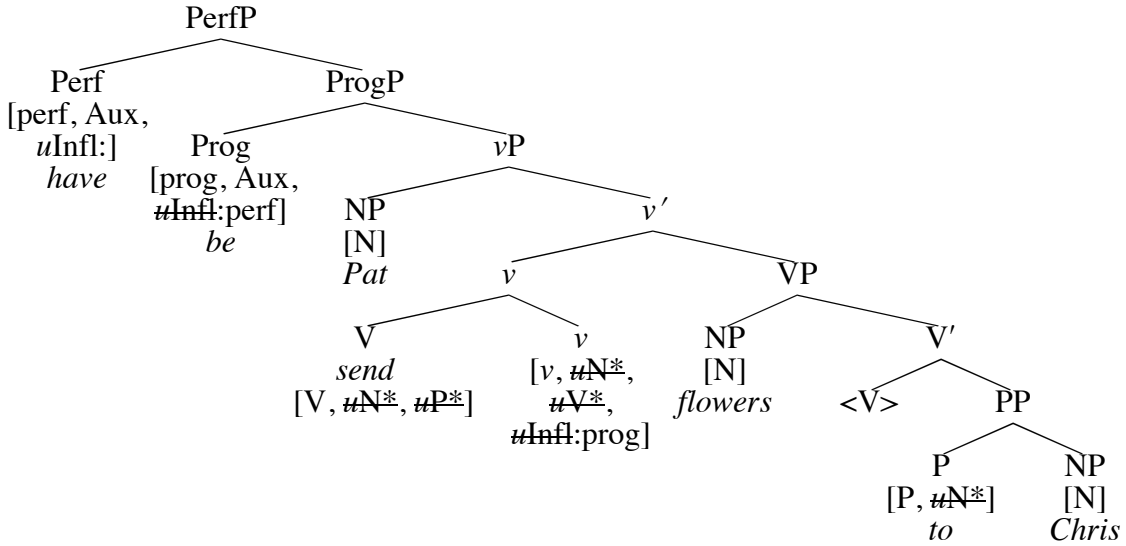
(This will ultimately sound like *"-ing"*—that is, V+*v* = "sending" when *v* has [~~*u*Infl~~:prog])

Because the feature is weak (Prog values the feature on *v*), the [*u*Infl:prog] feature is checked: [~~*u*Infl~~:prog].



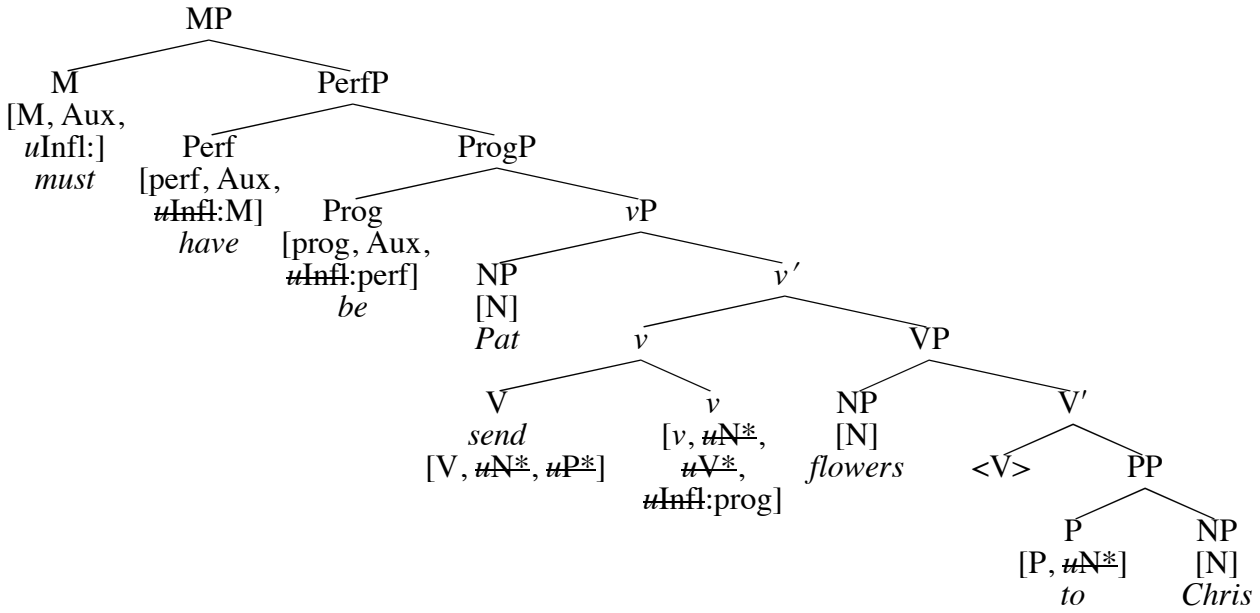
Step 8. Merge Perf and ProgP.

We are finished with Prog now, and Perf is the next step up on the HoP.
Because Perf is higher than Prog on the hierarchy of projections, features of Perf project.
Since Perf has no strong uninterpretable features to check, this is a maximal projection (PerfP).
The category feature of Perf ([perf]) can value an uninterpretable [*u*Infl:] feature.
Perf now c-commands Prog, which has a [*u*Infl:] feature still unvalued.
[perf] matches and values [*u*Infl:] on Prog resulting in [*u*Infl:perf] on Prog.
(This will ultimately sound like *"-en"*—that is, Prog = "been" when Prog has [~~*u*Infl~~:perf])
Because the feature is weak (Perf values the feature on Prog), the [*u*Infl:perf] feature is checked: [~~*u*Infl~~:perf].



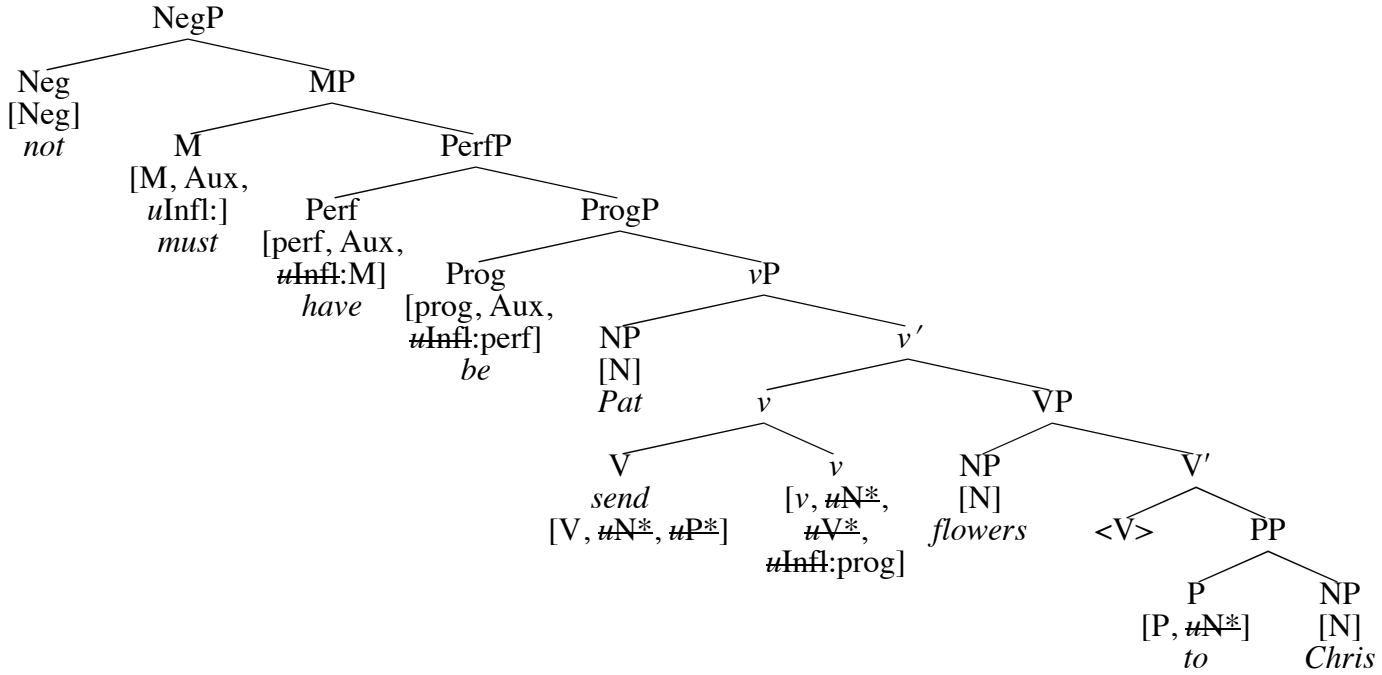
Step 9. Merge M and PerfP.

We are finished with Perf now, and M is the next step up on the HoP.
Because M is higher than Perf on the hierarchy of projections, features of M project.
Since M has no strong uninterpretable features to check, this is a maximal projection (MP).
The category feature of M ([M]) can value an uninterpretable [*u*Infl:] feature.
M now c-commands Perf, which has a [*u*Infl:] feature still unvalued.
[M] matches and values [*u*Infl:] on Perf resulting in [*u*Infl:M] on Perf.
(This will ultimately sound like *"-Ø"*—that is, Perf= "have" when Perf has [~~*u*Infl~~:M])
Because the feature is weak (M values the feature on Perf), the [*u*Infl:M] feature is checked: [~~*u*Infl~~:M].



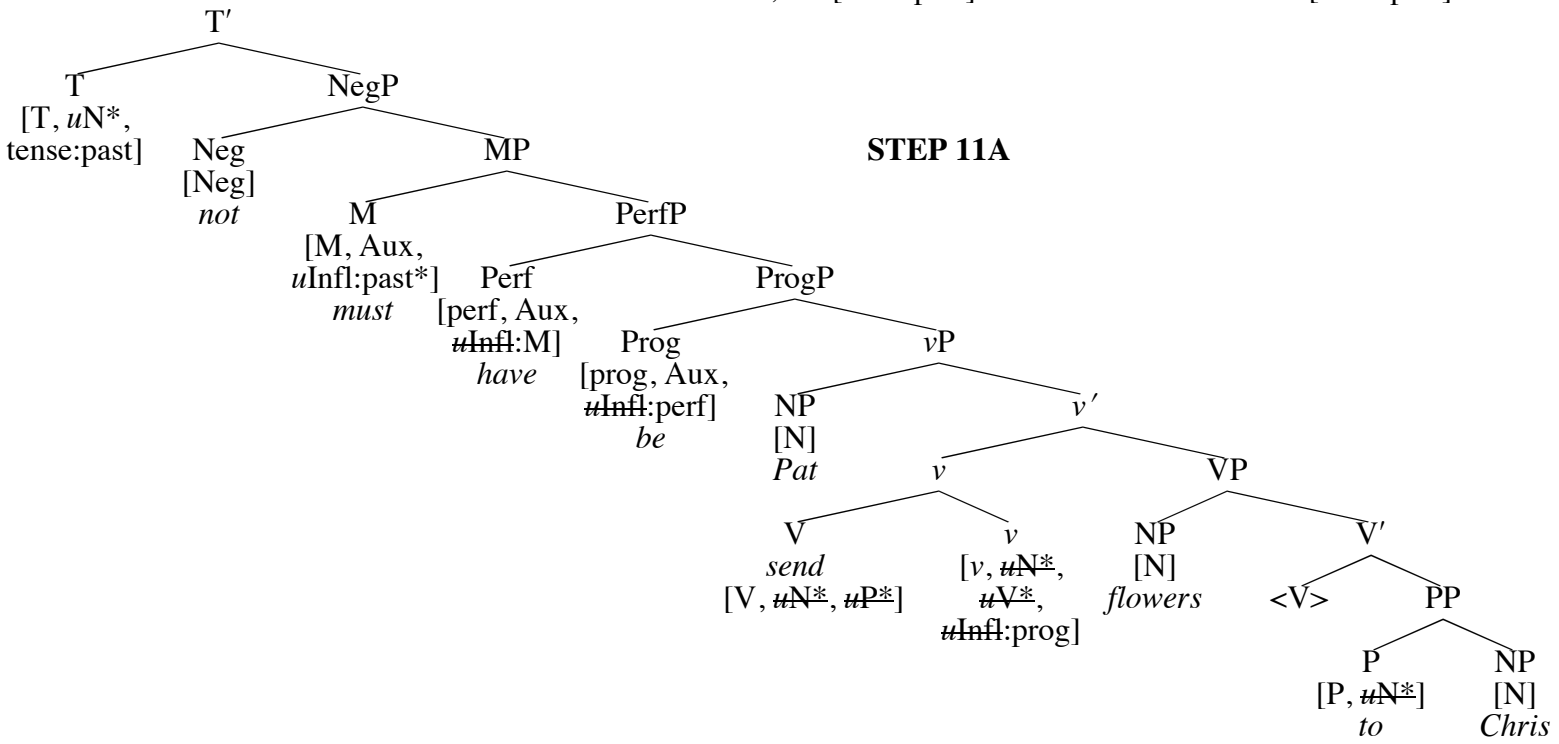
Step 10. Merge Neg and MP.

We are finished with M now, and Neg is the next step up on the HoP.
Because Neg is higher than M on the hierarchy of projections, features of Neg project.
Since Neg has no strong uninterpretable features to check, this is a maximal projection (NegP).
No features of Neg can value uninterpretable features, so nothing further to do in this step.



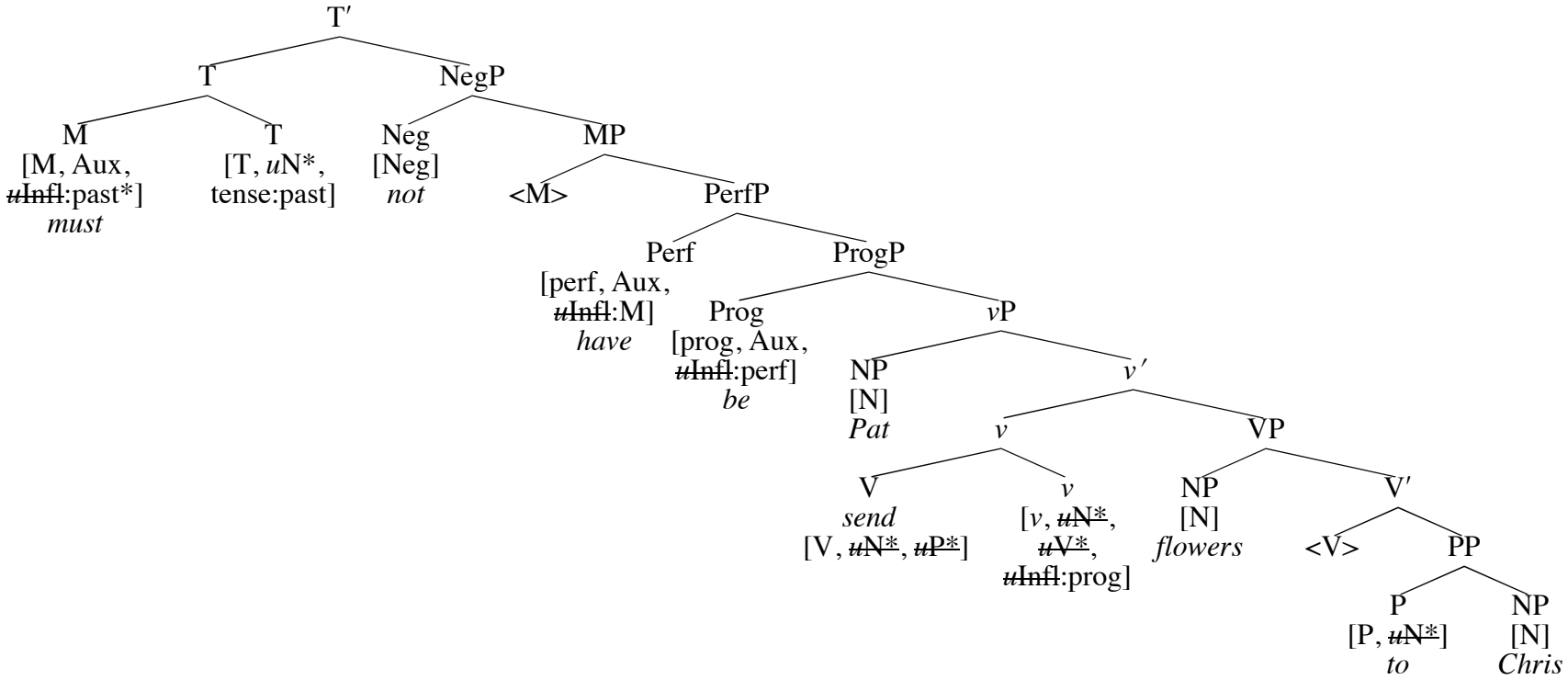
Step 11. Merge T and NegP.

We are finished with NegP now, and T is the next step up on the HoP.
Because T is higher than NegP on the hierarchy of projections, features of T project.
Since T has a strong uninterpretable feature to check, this is an intermediate projection (T′).
The [tense:past] feature of T can value an uninterpretable [*u*Infl:] feature.
T now c-commands M, which has a [*u*Infl:] feature still unvalued.
[tense:past] matches and values [*u*Infl:] on M resulting in [*u*Infl:past] on M.
(This will ultimately sound like *"must"*—that is, M= "must" when M has [~~*u*Infl~~:past])
This is the special case: when a [*u*Infl:] value on an auxiliary (M, Perf ,or Prog) is valued by T, the feature is strong, and can only be checked if it is local to T.
Thus, M must move to T (same procedure as moving V to *v*). **SEE STEP 11B.**
Once M has moved to T, the [*u*Infl:past] feature on M is checked: [~~*u*Infl~~:past].



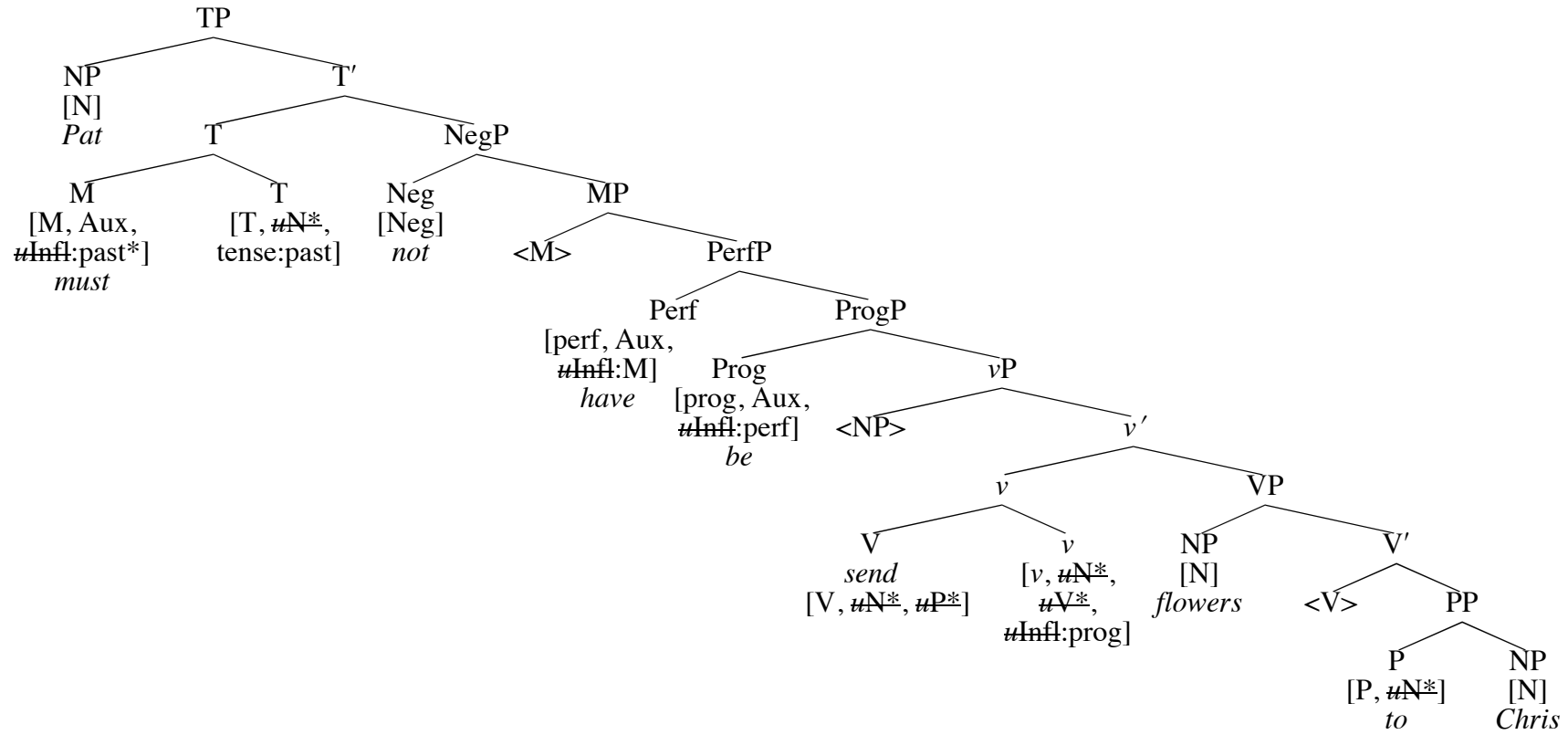
STEP 11A

STEP 11B:


T′
T
NegP
M
[M, Aux,
~~*u*Infl~~:past*]
must
T
[T, *u*N*,
tense:past]
Neg
[Neg]
not
MP
<M>
PerfP
Perf
[perf, Aux,
~~*u*Infl~~:M]
have
ProgP
Prog
[prog, Aux,
~~*u*Infl~~:perf]
be
*v*P
NP
[N]
Pat
v ′
v
VP
V
send
[V, ~~*u*N*~~, ~~*u*P*~~]
v
[*v*, ~~*u*N*~~,
~~*u*V*~~,
~~*u*Infl~~:prog]
NP
[N]
flowers
V′
<V>
PP
P
[P, ~~*u*N*~~]
to
NP
[N]
Chris

Step 12. Move *Pat* to the specifier of T.

T still has a strong uninterpretable [*u*N*] feature to check.
Nothing remains in the numeration that can check this feature.
The [*u*N*] feature of T matches the [N] feature of *Pat*. (T c-commands *Pat*, and there is nothing closer to T with an [N] feature).
Pat is moved (copied, and Merged with T′).
This puts the remaining [*u*N*] feature of T close to the [N] feature of *Pat*. [*u*N*] is checked.
Since T had its feature checked, the features of T project.
Since there are no more strong uninterpretable features on T, this is a maximal projection (TP).



The derivation is done. No uninterpretable features remain. Pronunciation: *Pat must not have been sending flowers to Chris*.