CAS LX 523 Syntax II
Spring 2001 March 27, 2001
Paul Hagstrom Week 9: *A Minimalist Program for Linguistic Theory—part two*

Chomsky (1998). *Minimalist inquiries: The framework*. Cambridge, MA: MIT Working Papers in Linguistics. Pages 27-32.

Chomsky (1995). Categories and transformations. In Chomsky, N., *The minimalist program*. Cambridge, MA: MIT Press. Sections 1–2.1, 3, 4.4 up to first third of p. 265, 5.1, 5.6 (just the small part on p. 297), 10.1

The strong minimalist thesis (Chomsky 1998)

(1) Language is an optimal solution to legibility conditions

We start with (1) and then we search for apparent imperfections—things that do not seem to arise from requirements of the interfaces.

One prominent example of this is *movement*. It seems to be an irreducible aspect of human language is that things appear to be *moved*—the appear in positions which are "displaced" from where they are interpreted (e.g. *What did you buy?*). This is different from other symbolic language-like systems, for example computer languages.

For any apparent imperfection P, any of the following could be true:

(2) a. P is a real imperfection. Therefore, (1) is false.
b. P is not real; the existence of P is only apparent, and (1) may still be true.
c. P is real, but not an imperfection—it is part of an optimal solution to the legibility conditions of the interfaces.

The goal is always (2b) or, better, (2c)—since (2c) give us insight into the legibility conditions themselves.

Some starting points, assuming (1) and the idea that less machinery is better than more.

(3) a. The only linguistically significant levels are the interface levels.
b. The **interpretability** condition: Lexical items have no features other than those interpreted at the interface (properties of sound and meaning).
c. The **inclusiveness** condition: The machinery of syntax (C_{HL}) does not introduce any new features not already contained in the lexical items.
d. Relations used by C_{HL} are either (i) imposed by the interface, or (ii) natural relations arising from the computational process itself.

Features (Chomsky 1995)

Features (really, "linguistic properties") are features *of* lexical items—lexical items have linguistic properties. Lexical items are bundles of features. For example, *airplane* has the property of being a noun—it is of category N. So *airplane* has the feature [N].

Syntax seems to care about features. At least, it seems to care about category features like [N], but it also seems to care about other features—gender, number, case, etc. Syntax seems to perhaps be *primarily* concerned with features and perhaps only epiphenomenally about the lexical items which host those features.

Features come in different kinds—linguistic properties have relevance to different things. There are **phonological features** which are the articulation-related properties of a lexical item. There are **semantic features** which are related to the meaning of a lexical item, to how that lexical item will be interpreted.

Since the computational component is providing representations for the interfaces (where representations are made of features, but organized in some way), we suppose that the phonological features are the things that the articulation interface cares about and that the semantic features are the things that the conceptual-intentional interface cares about. Moreover, since these are two very different systems, we can assume that they have no use for the features from the other interface. That is, **semantic features are interpretable as part of the LF representation**, and **phonological features are interpretable as part of the PF representation**. But, what is the phonological interpretation of a semantic feature? We assume that phonological features are enough to confuse the system interpreting LF, and that semantic features are enough to confuse the system interpreting PF. So, **semantic features are *uninterpretable* as part of the PF representation and phonological features are *uninterpretable* as part of the LF representation**.

Lexical items start their derivational journey with all of their features. How do we prevent uninterpretable features from getting to the interfaces?

Conclusion: **Spell-out** is an operation which **strips away phonological features and takes them to PF**. No phonological features remain to confuse the interpretation of LF, nothing but phonological features are taken to LF.

There is another distinction which seems to be necessary, among the LF-interpretable features. There are **formal features**, which are those features that are relevant to the operation of the syntax, and then there are **semantic features**, which are those features that are relevant to the interpretation of the meaning.

Chomsky's example: *airplane*:	[starts with a vowel]	phonological feature
	[artifact]	semantic feature
	[N]	formal feature

In general, semantic features (like phonological features) *do not* affect the way a lexical item is treated in the syntax. It is the fact that *airplane* is a noun, and not the fact that it is an artifact (vs. an abstract concept, vs., …) the determines where it appears and how it behaves in the syntax. Just like starting with a vowel doesn't affect whether something can, say, undergo *wh*-movement. The behavior in the syntax is the domain of the formal features.

It seems that the formal features of a lexical item are tied together in a way, that is (based on evidence we haven't looked at), they seem to travel around the syntax in a bunch. The "bag of formal features" is notated FF[α], that is the Formal Features contained in the lexical item α.

There is yet another distinction we can make between kinds of features, features which are **intrinsic** to a lexical item and those which are **optional**. In this discussion, we are primarily concerned with the formal features—the phonological and semantic features are presumed to be intrinsic. An **intrinsic feature** is one which "comes with the lexical item" no matter what structure we're looking at. For example, [masculine] on *he* (or [feminine] on French *lune* 'moon'). An **optional feature** is one which is chosen on a case-by-case basis, like [plural] or [Nominative Case].

Probably the most important distinction we have between formal features, though, is **interpretable** and **uninterpretable** (at the LF interface). Certain of the formal features (the features relevant to the syntax) also have an effect on the interpretation. One example of this would be the [N] feature indicating a noun. Another example of an interpretable feature is [plural]. Grammatical gender, however, does not seem to affect interpretation; gender is an uninterpretable feature.

The system must ensure that interpretable features reach the LF interface (so that they can be interpreted), while at the same time ensuring that uninterpretable features *do not* reach the LF interface (or they will confuse the system, the derivation will *crash*).

Checking features

Features which are uninterpretable at an interface must be eliminated before reaching that interface.

For phonological features, there is one way to eliminate them before reaching the LF interface: **Spell-out**. Spell-out strips away the phonological features from the representation and carries them off to PF. Notice that, given this, there's really no way to crash at PF in this system.

However—notice also that if you Spell-out too soon (say, before you've added a lexical item with phonological features to your tree), then there will be no way to get rid of those phonological features before the LF interface, and so the derivation will crash at LF. So you have to apply Spell-out *after* all of your lexical items are in place in the tree. This leaves open the possibility that a lexical item *without* any phonological features might be able to be added to the tree after Spell-out, incidentally.

Uninterpretable formal features are removed by *checking* them with other features. Here's a simple example:

Pat	[animate], [singular], [D], …
sings	[V], **[singular]**, …

[Singular] is interpretable on *Pat* but it is uninterpretable on *sing*—singular is the kind of thing that nouns are, not verbs. So, it's the *same feature*, but in one case it is in a position where it can be interpreted and in the other case it is in a position where it cannot.

The syntax brings *Pat* and *sings* together into a relation that is close enough (say, for now, a Spec-head relationship), and the uninterpretable [singular] on sings is destroyed by the contact with the interpretable [singular]. Something like matter and antimatter. The difference is: one (the interpretable one) survives (it *has* to—it's interpretable and needs to still be there when we get to the LF interface).

So, when a "checking relation" is established between two features (that is, they are brought in close proximity), the uninterpretable one(s) delete.

Note: Two uninterpretable features can check, upon which *both* delete. An example of this might be [Case] features. Two interpretable features can be placed in close proximity, but they are not considered to "check", just because there would be no point—neither needs to delete.

So, *interpretable features can check more than once*. A single interpretable feature on a noun can check multiple uninterpretable agreement features on verbs, auxiliaries, and participles. An uninterpretable feature is deleted right after it is checked.

Strong features

One thing that languages differ on is what movements in the syntax happen *before Spell-out*. Some languages move their *wh*-words to SpecCP before Spell-out, some wait until after Spell-out.

Chomsky (1993) had an answer to this that involved *strong* and *weak* features, and Chomsky (1995) retains this distinction but reformulates it in a particular way.

A [strong] feature is a feature which must be removed before Spell-out. A good way to ensure that this happens is to say it this way: **A [strong] feature is something which, once it has been introduced into the derivation, must be eliminated immediately**. Once you add a [strong] feature to the structure, the very next thing you do (more or less) has to be to eliminate it by bringing an appropriate feature in close proximity with it.

The particular formulation goes like this: If a head α has a strong feature, then that strong feature must have been eliminated by the time you *finish* building αP.

D is canceled if α is in a category not headed by α. (where α has a strong feature)

Bare phrase structure and the derivation

The inclusiveness condition (3c) says that our syntax can't add anything that wasn't already part of the lexical items. No new features. In particular, this also implies that we can't mark things as being an X-bar, or an XP, or a trace. We can't make syntactic use of indices (since they weren't part of the lexical item and because the syntax can't add anything). A phrase structure tree under this view must be simply lexical items, in combination: a *bare phrase structure*.

The derivation starts with a *numeration*. The *numeration* is a set of lexical choices (along with the number of times each item is to be used, e.g. *The dog bit the man*).

The computational system (recursively) generates syntactic objects from the objects available in the numeration and those syntactic objects already formed. We think of this as "rearrangement" of what we got in the numeration—putting things together, sometimes making copies, and that's it. Nothing added (the inclusiveness condition).

Given that we start with lexical items and need to end up with a single syntactic object, we need an operation that combines them. The simplest operation would be one that takes two existing objects and puts them together to form one—**Merge**.

Just from considering what in the tree is required by the interpretive component (semi-empirically), we find that properties of heads (minimal projections) and properties of phrases (maximal projections) are used. As it seems that these are the only things that are used, then we would want to limit our machinery in such a way that these are the only things which have conceptual status.

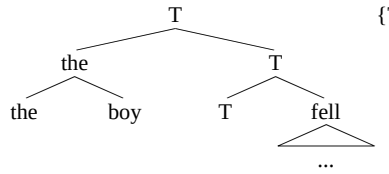
We need to be able to tell the difference between maximal and minimal projections, and we're not allowed to introduce any properties over and above those which were contained in the lexical items to begin with (inclusiveness), so we can't label maximal projections "P" and minimal projections "^{0}", for example. Instead, we'll have to tell if the thing we're looking at is an XP or an X^0 by considering its context in the tree.

Consider Merge. It takes two syntactic objects and puts them together. What kind of syntactic object are we left with? One possibility is that it is just the set of the two together, i.e. Merge(α, β) = {α, β}. This won't work, though, because verb phrases act like verb phrases and not noun phrases—we need the unit to somehow have the properties of the head of the phrase. We have to label the phrase, so the syntactic object formed has a label γ and the two merged objects α and β: {γ, {α, β}}.

Because we can't add anything (inclusiveness again), we know that γ must be something that we got from the lexicon, and since it is supposed to mark either α or β as the head, it is reasonable to say that γ is either α or β, depending on which one is the head of the complex (that is, depending on which one *projects*).

We can then read X-bar-like relations off of a tree:

(4)



{T, { {the, {the, boy}}, {T, {T, *fell* }}}}

A **head** is a terminal element drawn from the lexicon/numeration. The **complement** is the thing merged with the head, the most local relation. The **specifiers** are the other relations. The **maximal projection** of α is a constituent for which α doesn't project the label.

Notice the strange fact that *boy* up there is both a head and a maximal projection, all at once. This might possibly be useful for certain situations which seem at the same time both like head-movement and XP-movement (e.g., movement of a clitic from object position to adjoin to T).

Move F

Although trees seem to be built of lexical items put together and moved around, the syntax itself seems to care about *features*. The reason we move something from one place to another is to check features. The things which cause a derivation to crash are features. The things which tell us what lexical items are eligible to move are features.

Maybe we were looking at it the wrong way…

If the goal of syntax is to check features, then the simplest system would be one which finds the features that need to be eliminated and the features which can be eliminated and moves one to the other. And if that's the simplest way that the syntax could be designed, then it must be designed that way if (1) is right.

So why do we see whole words moving around? What's wrong with just moving a feature?

Chomsky blames an interface. In particular, he blames the PF interface. He supposes that it is prepared to pronounce words, bundles of features, but it is not prepared to pronounce bundles of features with "holes" in them. If you move a feature from one place to another to eliminate an uninterpretable feature, you have "scattered the features" of the lexical item across the tree, and the articulation system won't know how to pronounce it.

As a remedy to this problem, the computational system brings along the rest of the lexical item as well, when features are moved. It's forced by a property of the interface, so it is still an optimal solution to the requirements of the interface.

Specifically, suppose that the EPP is an uninterpretable [D] feature on T. All that needs to happen is that an interpretable [D] feature (from a noun/DP) moves up to T to check the uninterpretable [D] feature. However, this leaves a noun with a hole in it, so the next step is to "pied-pipe" the category containing the hole up to SpecTP, where the hole and the moved [D] feature are close enough to be pronounced together.

Notice also that because the PF interface is forcing this "pied-piping", nothing that happens after Spell-out needs to do this: **Covert movement is just feature movement, without pied-piping**.

This can also give some justification to the idea of "Procrastinate"—if moving before Spell-out means that not only do you have to move and check a feature, but you also have to pied-pipe the category containing that feature, then it is clearly more efficient to do your moving after Spell-out if you can.

Attract F

In fact, things become even a little bit simpler if we think of movement as "attraction" instead. That is, when a feature moves, it moves from lower in the tree to higher in the tree. Consider the EPP again. It says that "SpecTP must be filled (with a nominal element)". This is a property of T—it has some uninterpretable feature that must be checked off. That uninterpretable feature didn't even get into the derivation until we introduced T.

Moreover, if that feature is [strong], then we need to take care of it right away.

Under an "Attract" view, what happens is this: We introduce T into the tree and then look down into the tree for something that would be able to eliminate the uninterpretable feature. You only look until you find something, and once you find something, you move that. This gets **shortest move** effects, again following from a kind of least effort, but it gets it more directly. *Wh*-movement in English just looks down into the IP until it sees its first [wh] feature, and then moves that feature to C/SpecCP: Superiority.

The differences are a little bit subtle between a Move F and an Attract F approach, but Attract F feels more intuitive and is often the way minimalist discussions are cast.

There is no AgrP

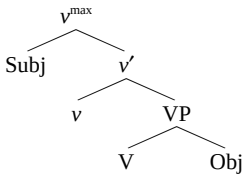
There are four main kinds of functional category (i.e. non-lexical category—N, V, Adj, and maybe P are lexical) that we've been dealing with: D, C, T, and Agr. But the first three have interpretive consequences. Agr seems only to be there in order to give us word order and agreement—it doesn't appear to have an interpretable features.

The way Agr works seems to be that a verb, with its uninterpretable ϕ-features (agreement) adjoins to the head, and then an argument, with its interpretable ϕ-features moves to SpecAgrP and checks off the uninterpretable ϕ-features on the verb. That is, Agr is only there to *mediate* feature checking.

Our only evidence for Agr seems to be from the cases where its features are strong—when there is overt movement to SpecAgrP. So, perhaps AgrP is only there when its features are strong. Can we say that AgrP is there just when it is strong?

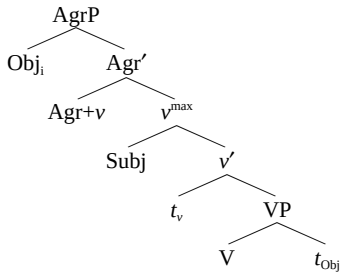
Consider object shift, starting here:

(5)



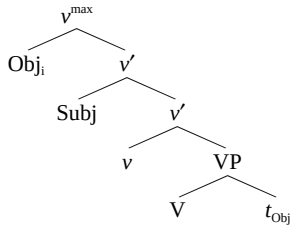
Shifting the object into SpecAgr would result in something like this (and we have overt evidence of this position in many cases): adding an Agr with a strong [D] feature to move the Object and a strong [V] feature to move the verb.

(6)



But couldn't we as easily just put the strong [D] feature on *v*? Even the subject has a [D] feature, it isn't in the checking domain of *v* for the simple reason that it was Merged there—the foot of even a trivial chain is not in the checking domain of the head it is merged with (this was discussed elsewhere in the chapter). So, instead, we have this, where the object just moves into a second specifier of v^{max} (still able to check the strong feature from there).

(7)



Assuming that adverbs can come between the shifted object and the subject, we can still maintain this structure, assuming that adverbs also occupy Spec*v*P and that the only requirement on strong features is that they be checked before leaving the projection (i.e. before merging *v*P with something else).

In the rare cases that really motivate AgrSP (multiple subject constructions, like in Icelandic, where you can have an expletive *and* a subject at the beginning of the sentence), we can do the same thing, assume that T has two specifiers in that case.

A couple of notes

In coming up with this system, Chomsky has intentionally stuck to the simple cases, and has swept no small amount of complication under the rug. For example, in those MSC constructions, the verb usually appears in *second position*, which seems at odds with considering both the expletive (preverbal) and the subject (postverbal) from being in specifiers of T. One suggestion he made the following year has this being more a matter of phonology than of syntax (suggesting that "second position" was not a syntactically defined position but a prosodically defined one, and therefore suggesting something like [EXP SUBJ T… as the structure of the syntactic tree anyway). In fact, he went so far as to say that head movement itself wasn't a property of the syntax, only XP movement—we can talk about this more if there's time. Other more recent developments include "multiple-spell-out" and "phases" which we can talk more about if there's time.