The enterprise (reminder):

- Language faculty, part of being human (genetically specific).
- Comprises a system which highly restricts the range of possible human languages.
- This system underlies all human languages, we seek to discover its properties.
  - Looking in depth at the syntactic properties of individual languages
  - Comparing properties across languages to see what the options are
- Once we see patterns, we
  - state generalizations
  - form a hypothesis (what causes the generalization?)
  - check other predictions of the hypothesis
  - revise as needed
- As the theoretical hypotheses develop, we see new things to check that we wouldn’t have thought to check before. In the process, whether the hypotheses are right or wrong, we have learned a lot about how language works.
- **We assume there is a fact of the matter, that the Truth is Out There.**
  We are just trying to figure out what it is.
- Lots of proposals exist, some probably closer to the Truth than others.
- To evaluate them, we must see what they predict:
  - Do they predict that all of the possible languages/constructions exist? 
    *English sentences must be SVO.*
    I left. Taxes I can deal with, it’s the forms that bother me.
  - Do they predict that all of the impossible languages/constructions don’t exist? *English sentences may have any word order at all.*
    *Like I pizza. *Bill thinks Mary that left.
- When two hypotheses both seem to predict the data, we get into aesthetics.
  - Which hypotheses are “simpler”, more “elegant”? 
  - Which hypotheses make the greatest range of predictions?

(1) CP

The clause structure of “the olden days” (say, 1986):

- CP: force (declarative, interrogative)
- IP: functional morphology (tense, agreement)
- VP: lexical/thematic elements

But this doesn’t seem to be quite good enough to describe/explain the data—progressively, they have been split up into further functional projections.
More modern clauses tend to look more like this (at least):

(2) ForceP
    /   \
   /     \  "CP"
TopP*   FocP
    /   \    
   /     \    
TopP*   FinP
    /   \    
   /     \    
AgrSP   TP
    /   \    
   /     \    
NegP   AgrOP
    /   \    
   /     \    
vP     "VP"
    /   \    
   /     \    
VP

A little bit of history

This kind of generative grammar has progressed through several stages, sometimes names.

Aspects
Standard Theory
Extended Standard Theory (EST)
Revised, Extended Standard Theory (REST)
Government & Binding (GB)
    Lectures on Government and Binding (1981)
    Barriers (1986)
(Kayne 1994: Antisymmetry)
Minimalist Program (MP) (sort of)
    A Minimalist Program for Linguistic Theory (1993)
    Chapter Four (1995)
    Derivation by Phase (1999)
    Beyond Explanatory Adequacy (2001)
There have also been offshoots along the way (again, sort of)
Lexical Functional Grammar (LFG)
Head-driven Phrase Structure Grammar (HPSG)
Relational Grammar (RG)

Background to the Minimalist Program

CHL, stuck between A–P (PF) and C–I (LF), which impose requirements.
Is human language more than an optimal solution to those requirements?
Is all of the complexity we find in syntax ultimately traceable to the design requirements?

Strong Minimalist Thesis:
(3) Language is an optimal solution to legibility conditions

Some starting points, assuming (3) and the idea that less machinery is better than more.

(4) 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 (CHL) does not introduce any new features not already contained in the lexical items.
d. Relations used by CHL are either (i) imposed by the interface, or (ii) natural relations arising from the computational process itself.

Features and the lexicon—Part one

UG makes available a set F of features (linguistic properties) and computational operations CHL (the computational procedure for human language) that make use of F to form expressions. A particular language L uses F to form a particular set of expressions {EXP}. A given EXP is a pairing of meaning (LF) and sound (PF), interface representations.

A language L takes those features and bundles them, forming a lexicon.

Bare phrase structure

(4c) says that CHL can’t add anything that wasn’t already part of the lexical items. No new features.
This implies that we can’t (meaningfully) 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**, which is an array of lexical choices to be reorganized by the system. (Not a set, but close).

The computational system (recursively) generates **syntactic objects** from the objects available in the numeration and those syntactic objects already formed. We “rearrange” what we had in the numeration—putting things together, sometimes making copies, and that’s it. Nothing added.

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

What kind of syntactic object do we get after Merge? Must be made of things we already had in the lexicon (bundles of features). Merge([a], [b]) = {[a], [b]}? No, verb phrases act like verb phrases and not noun phrases—we need the combined object to have the properties of the head of the phrase. We have to label the phrase to indicate what kind of phrase it is.

The syntactic object formed by Merge has a label [g] and the two merged objects [a] and [b]: {[g] {a, b}}.

We can’t add anything, so [g] must be something based on either [a] or [b]; since it is marking which of [a] or [b] is the head, we can just say that [g] is either [a] or [b], depending on which one is the head of the complex (that is, depending on which one projects).

![Diagram](https://via.placeholder.com/150)

We can then read X-bar-like relations off of a tree, just by looking at the context.
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 [] is a constituent for which [] doesn’t project the label.

What this means is that Merge derives much of what X-bar theory stipulated.
- binary branching (Merge combines two things)
- one head
- properties of the head determine the properties of the phrase
- complements and specifiers must be maximal projections

But there are some things left over:
- can only have one specifier?
- can only have one complement?
- must have three levels (head, intermediate, maximal)

Also notice that cradle is both a head and a maximal projection, all at once.

Question is: Were the extra parts of X-bar theory really doing us any good? Can we do without the parts of X-bar theory that don’t come for free?

(for free … derives from Merge, which we an interpretable language needs anyway… LF only interprets single syntactic objects, lexicon provides a bunch of objects in the numeration, so we have to combine them…)

Procedure: Start with (3), then seek to disprove it. Search for apparent imperfections—things that do not seem to arise from requirements of the interfaces.

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

(6)  
   a. P is a real imperfection. Therefore, (3) is false.
   b. P is not real; the existence of P is only apparent, and (3) 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 (6b) or, better, (6c)—since (6c) give us insight into the legibility conditions themselves.
As you were

Now that we have a conceptual basis for phrase structure, we can go back to writing things pretty much how we did before, just with these issues in the back of our minds.

(7) \[ \begin{array}{c}
\text{T} \\
\{ \text{will}, \{\{\text{the,}\{\text{the,cradle}\}\}, \{\text{will,}\{\text{will,fall}\}\}\}\} \\
\end{array} \]

\[ \begin{array}{c}
\text{D} \\
\{ \text{cradle} \}
\end{array} \]

\[ \begin{array}{c}
\text{will} \\
\{ \text{will} \}
\end{array} \]

\[ \begin{array}{c}
\text{V}
\end{array} \]

(detail still missing in “VP”)

This is just labeling the nodes with the category feature (arguably one of the most important features of a lexical item) rather than spelling it all the way out.

We could even go back to writing in the bars and Ps, just with the understanding that they have no meaning for the syntax (and we don’t write any “empty” things, so no non-branching nodes, etc.). People often do this, just for comfort.

(8) \[ \begin{array}{c}
\text{TP} \\
\{ \text{will}, \{\{\text{the,}\{\text{the,cradle}\}\}, \{\text{will,}\{\text{will,fall}\}\}\}\} \\
\end{array} \]

\[ \begin{array}{c}
\text{D} \\
\{ \text{the} \}
\end{array} \]

\[ \begin{array}{c}
\text{NP} \\
\{ \text{cradle} \}
\end{array} \]

\[ \begin{array}{c}
\text{T} \\
\{ \text{will} \}
\end{array} \]

\[ \begin{array}{c}
\text{VP} \\
\{ \text{will,fall} \}
\end{array} \]

What syntax cares about

Syntax cares about some features more than others. It cares about category. It doesn’t care about “starts with p”. No known language treats two sentences, one with cat and one with dog, differently in any formal way.

We could even go so far as to say that syntax doesn’t really “see” anything but certain features (even if there are other features in the “bundle”). It sees category features ([T], [D], etc.), and so it deals only with those.

(9) \[ \begin{array}{c}
\text{T} \\
\{ \text{T}, \{\{\text{D,}\{\text{D,N}\}\}, \{\text{T,}\{\text{T,V}\}\}\}\}\} \\
\end{array} \]

\[ \begin{array}{c}
\text{D} \\
\{ \text{D} \}
\end{array} \]

\[ \begin{array}{c}
\text{N} \\
\{ \text{D,N} \}
\end{array} \]

\[ \begin{array}{c}
\text{T} \\
\{ \text{T} \}
\end{array} \]

\[ \begin{array}{c}
\text{V}
\end{array} \]
Same operations for *The cradle will fall* and *A boat might sink*. No different as far as syntax is concerned.

C-command and other relations

With BPS, we can define certain syntactic relations quite easily.

**Sister of** (other element in a Merge)

**Contain** (sub-elements of a syntactic object)

This also gives us **c-command**, which has proven to be very important in syntax.

**Contain** (**Sister of**)  

One relation that we don’t seem to have a very easy time with is **government**.

*Recall*:

X governs its sister, its specifier (under some definitions), and the specifier of its sister.

![Diagram](https://via.placeholder.com/150)

We can’t even really talk about the specifier relation in any simple way (we’ll get back to that), but it’s *really* hard to talk about specifier of the sister in a simple way.

Accusative case assignment was for a long time considered to require the relation of government.

In a BPS system, we can’t make use of that relation—we need another way to do Case. We’ll return to that.

Movement

It is an irreducible fact about language that things *move* from the place where they’re interpreted.

(11) Bill is likely to seem unhappy.
What did Mary ask Sue to buy?

The sandwich was eaten by Martin.

Putting aside why for a moment, how do we move things?

Prior syntactic research has given us reason to believe that the subject moves from SpecVP to SpecTP.

(14) workspace: \{Bill, [PAST], saw, Mary\}

combine saw and Mary:

\[
\begin{array}{c}
V \\
\text{saw} \\
\text{DP} \\
\text{Mary}
\end{array}
\]

(15) workspace: \{Bill, [PAST], [V\text{\textordertext}`\text{\textordertext}saw Mary]\}

combine Bill and [V\text{\textordertext}`\text{\textordertext}saw Mary]::

\[
\begin{array}{c}
\text{VP} \\
\text{DP} \\
\text{Bill} \\
V \\
\text{saw} \\
\text{DP} \\
\text{Mary}
\end{array}
\]

(16) workspace: \{[PAST], [VP Bill saw Mary]\}

combine [PAST] and VP:

\[
\begin{array}{c}
T \\
\text{[PAST]} \\
\text{DP} \\
\text{Bill} \\
V \\
\text{saw} \\
\text{DP} \\
\text{Mary}
\end{array}
\]

(17) workspace: \{[TP [PAST] Bill saw Mary]\}

Now what? We have nothing left. (Well, maybe C).

What do we want to end up with?

**Idea:** Instead of Merge of two objects from the numeration, we pick something inside the tree, make a copy of it, and Merge it with the tree.
Chains and traces

Traces are now just copies of the moved element. So how do we know which one to pronounce?

If two identical elements are in a c-command relationship, they are part of a chain. We pronounce the top one—we pronounce the one that isn’t c-commanded by an identical element.

Perhaps not obviously a requirement of the PF interface, but it is at least statable in BPS.

This also means that traces are copies. Everything that was in the original element is there (unpronounced) in the base position. This too is different from the way people used to think about it—but it gives us an explanation of some reconstruction facts.

Among the simpler cases of this:

Which song about himself did Mick say that Keith likes?

The pertinent point here is that himself can be interpreted either as Mick or as Keith.

Mick [past] say that Keith likes which song about himself.

Mick [past] say which song about himself that Keith likes t.

which song about himself [past] Mick t say t that Keith likes t.

Then we get the readings by putting the wh-phrase back somewhere between SS and LF. And you can only put it back into a place where it has once been.
The *copy theory of movement* resolves this in a way which seems to make more sense.

*Mick [past] say that Keith likes wh.s.a.self.*

*Mick [past] say wh.s.a.self that Keith likes wh.s.a.self*

*wh.s.a.self [past] Mick [past] say wh.s.a.self that Keith likes wh.s.a.self*

If there are copies of this in each place, we might suppose that LF can pick one (of the members of a chain) to interpret—and it need not be the top one. It need not be the one that PF picked to pronounce.

---

**Movement and features**

There are two prominent “imperfections” in human language, as we understand it.

- Lexical items have uninterpretable features
  - Agreement—doesn’t add anything to the meaning
- Things move away from the position in which they are interpreted.
  - *What did the students all buy? Why move what? Why move the students?*

If we were designing a language, it wouldn’t seem like we would include either of these. Cf. computer languages.

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

(19)  
  a. $P$ is a real imperfection. Therefore, (3) is false.  
  b. $P$ is not real; the existence of $P$ is only apparent, and (3) 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.

Perhaps we can do one better if we can at least cut down our two imperfections to one.

Chomsky’s idea: Movement is necessary. (the remaining imperfection?)

Given that, uninterpretable features are there to *effect* movement.

(could even be an optimal solution to the problem of movement)

So, we may have reduced the existence of uninterpretable features to a case of (19c).
The way it works:

- Lexical items enter the numeration with uninterpretable features.
- Uninterpretable features are just that—they can’t be interpreted at the interfaces.
- They must therefore be eliminated between the numeration and the interface.

We need a way to eliminate uninterpretable features. Think about agreement. We observe that agreement is a relation between two things (a V and an N, say).

<table>
<thead>
<tr>
<th>Pat</th>
<th>[animate], [singular], [D], …</th>
<th>[singular] is interpretable here.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sings</td>
<td>[V], [singular], …</td>
<td>[singular] is uninterpretable here.</td>
</tr>
</tbody>
</table>

Why agreement? Why do the uninterpretable features that do show up seem to be the same as those on some other lexical item?

This suggests that what eliminates uninterpretable features is checking. An uninterpretable feature can be eliminated by finding another feature like it and “getting them together” somehow.

Since checking uninterpretable features is supposed to be the implementation for movement, the natural suggestion is that checking occurs only in a local relation. You have to get the uninterpretable feature close to the paired feature it will check with.

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: the interpretable one survives (it has to—it’s interpretable and needs to still be there when we get to the LF interface).

**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. Checking is something that happens to uninterpretable features.

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.
As for why we need movement at all (the remaining imperfection), we’ll talk later. Chomsky has a speculation about that which basically centers on the fact that human language is involved in discourse (connected utterances), which sets it apart from computer languages.

**Least effort**

When looked at closely, movement seems to have a least effort character. Often, you find that a movement will not happen if there was a shorter movement that could have accomplished the same thing.

Evidence for **Attract Closest**: Superiority in English, Raising in Icelandic.

(20) a. Who, did John say $t_i$ bought what?

\[ \text{a----------l} \]

\[ \text{(highest wh-word must move)} \]

\[
\begin{array}{c}
\downarrow \\
(20) \quad \text{Who, did John say } t_i \text{ bought what?} \\
\downarrow \\
\end{array}
\]

b. * What, did John say who bought $t_i$?

\[ \text{a----------=-----l} \]

\[ \text{b. * Who} \]

\[ \text{b. * What} \]

(21) a. Jón telur [mér, virðast $t_i$ Haraldur hafa gert þetta vel]. J.NOM believes me.DAT to.seem H.NOM to.have done this well

‘Jon believes Harald to seem to me to have done this well.’

\[ \text{b. * Jón telur [Harald, virðast mér $t_i$ hafa gert þetta vel].} \]

J.NOM believes H.ACC to.seem me.DAT to.have done this well

‘Jon believes Harald to seem to me to have done this well.’

\[ \text{b. * Jón telur [Harald, virðast mér $t_i$ hafa gert þetta vel].} \]

J.NOM believes H.ACC to.seem me.DAT to.have done this well

‘Jon believes Harald to seem to me to have done this well.’

\[ \text{c. Jón telur [ Harald, virðast $t_i$ hafa gert þetta vel ].} \]

J.NOM believes H.ACC to.seem to.have done this well

‘Jon believes Harald to seem to have done this well.’

(22) a. * Have, John will $t_i$ left by the time we get there?

b. * John, is likely for it to seem $t_i$ to have left.
This suggests that the syntactic computation takes the *cheapest way out*. Longer moves are somehow more expensive, and thus avoided.

This was originally known as **Shortest Move**. Movement is as short as it can be.

The current view of this is actually the other way around—**Attract Closest**.

Consider the EPP. SpecTP must be filled with a DP.

We take this to mean that T has an uninterpretable feature that a DP has. Simplest way to view this is that T has an uninterpretable [D] feature.

To eliminate the [D] feature, we need to get a DP in close proximity—we need to move one to SpecTP. And it has to be the closest one.

(23) Bill ate the sandwich.
(24) *The sandwich did Bill eat.

So, we’ve built up to here:

(25)

```
T
   [D], [PAST]
   VP
   DP
     Bill
   V
     ate
   DP
     the
     sandwich
```

At this point we need to find (or, *attract*) a DP to solve the [D] problem on T.

If you start looking down into the object you have…

```
{T, {T, {ate, {Bill, {ate, {ate, {the, {the, sandwich }}}}}}}}
```

…the first D you come to is **Bill** (not **the**).

The computational system doesn’t look further than it has to. It finds the closest D (**Bill**), and moves it. That is, it copies it, and merges it onto the object. The D feature is checked, the representation is interpretable (at least with respect to this part).
Case and government

We allow complement and specifier relations—local relations provide directly by the phrase structure. No more—no government.

Case? Can’t do object and subject Case differently anymore, we must unify Case assignment to Spec-head relations, which we can do in the spirit of Pollock’s Split-INFL:

Chomsky (1993) suggested that nominative Case is assigned (checked) in SpecAgrSP and objective/accusative Case is assigned in SpecAgrOP.

Since objective Case depends on properties of V and nominative Case depends on properties of T, we might assume that T raises to AgrS and V raises to AgrO to put these features in local relations with the NPs that receive Case in the specifiers.
So, subject and object Case assignment is very symmetrical:

- The subject starts in SpecVP and moves to SpecAgrSP
- T moves to AgrS
- The object starts as the complement of V and moves to SpecAgrOP
- V moves to AgrO

Adjective agreement can be done the same way, with the adjective raising to AgrA, and the subject raising to SpecAgrA (and then perhaps further to SpecAgrOP in *I consider Bill intelligent*, or to SpecAgrSP in *Bill is intelligent*).

**Language variation and Spell-out**

So far, there is nothing in the system that allows languages to differ except in the lexical items that come in from the numeration.

If we stick to the minimal necessary assumptions, that’s the only option we have.

**Language variation is restricted to the lexicon.**

The computational procedure (Merge, Move, … C_HL) is the same for all languages.

Another thing we haven’t considered: Lexical items have phonological features too (“starts with p”) but these are assumed to be uninterpretable at LF. So they must be removed.

Phonological features are not checked. Instead, they are eliminated all at once by an operation called Spell-out. Spell-out removes the phonological features and sends them off to the PF interface. It’s similar to the breakpoint in the old-style system called SS.

When do we do Spell-out?

The least effort intuition (combined with forethought) suggested that Spell-out happens as early as possible, under the assumption that doing movement you can hear is more work than doing movement that you can’t. This is sometimes referred to as Procrastinate.

The computational system would prefer to send everything off to phonology immediately.

Why doesn’t it?
One reason is that Spell-out is the only way to eliminate phonological features. **So, you can’t Spell-out before you’ve added all of the lexical items to your tree** (except maybe for special lexical items that have no phonological features) because otherwise you would have no way to get rid of them. The tree would arrive at LF with phonological features and the derivation would **crash**.

In English, we can see accusative Case and that the verb doesn’t move to T:

> Bill often calls me.

Since we don’t see the object move, but it does get Case and so must abstractly move to SpecAgrOP, we conclude that the object only moves after Spell-out. In general, languages seem to differ in just this sort of way (other languages do seem to move objects to SpecAgrOP overtly, at least sometimes). How do we capture that?

Certain uninterpretable features are special. They are **strong**. A strong feature must be eliminated prior to Spell-out. Which features are strong may vary from language to language.

The EPP is a strong uninterpretable [D] feature on T. The [D] feature on T must be removed before Spell-out.

So, Spell-out happens as soon as it can, but that’s going to be only after all of the strong features are eliminated.

In French, the verb raises overtly to T. In English it doesn’t.

This can be characterized in this system by saying that French has a **strong** V feature on T. In English, assuming there is also an uninterpretable V feature on T, it is **weak** and so V doesn’t move to T until after Spell-out (when we can’t hear it).

A **strong** feature is something like a “virus”—if it gets into the computation, it must be eliminated as soon as possible. You can’t wait. In practice, this means that a strong feature must be eliminated by the last stage at which the head containing the feature projects. If a feature is **weak**, then we Spell-out first, and then move the object to SpecAgrOP.