Overview of the Minimalist Program (Marantz 1995)

A fairly major change in thinking between GB and the Minimalist Program. Before a
derivation just did whatever it does (Move-\(\alpha\) applies freely), and succeeds so long as the
result meets the conditions imposed by the various sub-modules (Case Theory, \(\theta\)-theory,
Binding Theory, etc.). In the MP, each movement (or operation) has a motivation—the
theory becomes much more closely tied to the derivation.

Partly this comes from things which seem to have a “least effort” flavor—you can only
move the closest relevant entity, which seems to suggests that “moves need to be as short
as they can be.” A longer move loses out to a shorter move. This also introduces a notion
of comparison between possible derivations.

Shortest Move: A constituent must move to the first position of the right kind up from its
source position.

This covers the head movement constraint (you can’t move a head past another
head—since you could have moved it to the closer head), Superiority (you can’t move a
further wh-word if there is a closer wh-word), super-raising (you can’t move a subject past
a possible subject position)

(1) a. * Have \(i\) John \(\text{will}\) \(ti\) left by the time we get there?
b. * John \(i\) is likely for \(it\) to seem \(ti\) to have left.
c. * What \(i\) did you persuade \(who\) to buy \(ti\) ?

Determining what the “comparison set” is becomes complex—the examples in (1) are bad
because there is a competitor in the comparison set which fares better with respect to

Shortest Move. But we can’t have the derivation of I left competing with everything, since
it would almost always win—rather, we need to limit comparison to derivations involving
the same lexical items and the same interpretation.

The structure of the derivation in the GB model:

(2) \[
\begin{array}{c}
\text{DS} \\
\downarrow \\
\text{SS} \\
\downarrow \\
\text{PF} \\
\end{array}
\]

Under the Minimalist Program, there are only the interface levels (no conditions apply to
other representations along the way); there is no DS or SS. There is just the initial set of
lexical items and the computation, which at some point (“Spell-out”), splits off and hands
off the current state to be pronounced.

(3) \[
\begin{array}{c}
\text{lexical array} \\
\leftarrow \text{Spell-out} \\
\downarrow \\
\text{PF} \\
\text{LF} \\
\end{array}
\]

In the same spirit of “least effort,” supposing that overt movement is more costly than covert movement, there is a preference for movement to occur after Spell-out (covertly).
This is referred to as Procrastinate (as in: “hold off movement until after Spell-out”),
although we can equivalently think of it as doing Spell-out as early as possible.

The tree is the same shape, but we can no longer state conditions that must hold of DS or
of SS.

Movement generally arises for reasons of “feature checking” which has the effect of
removing features from lexical items. These features come in two varieties, strong
features and weak features. A strong feature is considered to be a feature (uninterpretable
to the articulation interface) which is visible to PF and therefore must be eliminated before
reaching Spell-out. A weak features is not visible to PF and can therefore be left until after
Spell-out. In practical terms, a strong feature drives overt movement, while a weak
feature allows covert movement (and in light of Procrastinate, usually forces covert
movement).

A passing mention of **Greed**: The principle is supposed to ensure that an NP can’t move
just to satisfy the requirements of some other lexical item, but rather must move only if it
needs something. This principle is fairly problematic and was dispensed with in Chomsky
(1995) when movement was re-interpreted in a way we’ll get to.

Derivations, deep structure, and the starting point: If the interface levels are all we have,
then we don’t have DS anymore as the starting point of the derivation. All we have is the
list of lexical items that we’re going to use in constructing the representations at the
interface.

In the MP grammar, trees are built from the bottom up by taking lexical items from that
list and combining them with each other and with structures built previously in the
derivation (or in some cases, moving them from inside a structure already built to the top).
That is, things that are lower in the tree happen first in the derivation.

Marantz’s metaphor for the derivation envisions the lexical array laid out on a workbench,
and the grammar’s task to be to take items on the workbench and combine them into
composite items, which can then be combined with other items until there are no stray
pieces left.

Movement is seen as basically picking some internal piece of the tree we’re working on
and attaching a copy of it to the outside of the tree (but we don’t change anything in the
original tree itself—in particular, there are no longer traces of movement; what’s left in
the original position is [the original] copy of the thing moved).

A requirement on derivations is that any operation on a syntactic element must extend
it—it must be the original element plus something. This prevents moving something from
inside a tree to inside a tree later in the derivation—movement is always to the top of the
tree you are constructing at the moment. This is the MP realization of a concept called
“cyclicity” (which basically ensures that operations occur on internal constituents before it
occurs on containing constituents—a property of grammar that has been assumed for a
long time, actually with roots in derivational phonology).

Marantz points out that causes a bit of a problem for head movement, since head-
movement does not have that property, and there are other problems involved in covert
movement as well. This is another area which is addressed in later implementations of MP.

The “lexicalist hypothesis” that is part of the MP way of looking at things deviates from
previous approaches in its handling of morphological affixes. In GB analyses, it was often
assumed that the affixes (like English –ed) are inserted as the head of, e.g., Tense. In MP,
this is not assumed. Rather, it is assumed that the lexical items that form the initial lexical
array include a fully formed verb (like walked) and an abstract T node with the feature
[PAST]. Because the verb has a [PAST] feature (by virtue of it having entered the derivation
as walked), it will need to enter a local relation with T at some point in the derivation so
that the [PAST] feature on the verb can be “checked” against the [PAST] feature of T. So
long as the verb and T agree on this feature, the derivation is fine.

This “feature checking” is generalized beyond this to be the general motivation for
movement. Where the verb moves up to T, it has a strong [V] feature which needs to be
checked (before Spell-out) by the verb. Feature checking can take place either between a
head and something adjoined to that head, or between a head and its specifier.

An example of a derivation of something like **John saw Mary** (ignoring AgrOP & AgrSP)

(4) workspace: [John, [PAST], saw, Mary]

    combine saw and Mary:
    \[ V' \]
    \[ V \]
    \[ saw \]
    \[ DP \]
    \[ Mary \]

(5) workspace: [John, [PAST], [V′ saw Mary]]

    combine John and [V′ saw Mary]:
    \[ VP \]
    \[ DP \]
    \[ John \]
    \[ V' \]
    \[ saw \]
    \[ DP \]
    \[ Mary \]

(6) workspace: [ [PAST], [VP John saw Mary]]

    combine [PAST] and VP:
    \[ T' \]
    \[ VP \]
    \[ DP \]
    \[ John \]
    \[ V' \]
    \[ saw \]
    \[ DP \]
    \[ Mary \]
The “Minimalist Program” is to try to address that question. Slightly more practically, it is an attempt to see if the answer is: “FL is an optimal solution to the design specifications.” To what extent do minimal conditions of adequacy suffice to determine the nature of the right theory?

The “therapeutic” value of the MP is to focus our attention on what stipulations are needed beyond what is “conceptually necessary” for a language system—in a way which people have not been used to doing previously.

2. Design specifications

To get started, we need to have some idea what a “good design” is, as well as what the minimal design specification for a language system would be.

For “good design” we have to, for the moment anyway, resign ourselves to “we know it when we see it.” There are lots of very good ways to measure a design’s “goodness” and they often go in different directions.

To communicate with the articulatory system and the thought system, the representations at the interface must be legible (interpretable) by the system. They must be in some sense be “in the vocabulary” of the system on the other side.

Terminology: A PF representation legible to the articulatory system converges at PF. A LF representation legible to the interpretation system converges at LF. An expression (or the computation leading to the expression), with both a PF and LF representation, each of which converges, will itself be said to converge.

Certain features (=properties) of lexical items are interpretable at one or the other interface (i.e. interpretable to the outside system); other features of lexical items are uninterpretable. We will assume that if an expression contains only features interpretable at the respective interfaces, it will be among the convergent expressions.

Footnote about intelligibility: A convergent representation may not make sense—but it is “readable” by the outside system (which can then make a decision of its own as to how sensible it may be).

1. Background

There is a human faculty of language, a component of the mind/brain dedicated to language. We suppose that the language L provides information to the performance systems in the form of representations which the performance systems access. We’ll assume that there are at least the sensorimotor system (responsible for articulation and perception) and the system of thought—and we’ll take these two to be separate, accessing different representations. The language L, then, is a system to provide two representations (or, together, an “expression”), one for the articulation system and one for the interpretation system. To attach familiar names, PF (“instructions” for the articulatory system) and LF (“instructions” for the interpretation system).

We have some experience now with grammars under the Principles & Parameters approach, and many things have been learned. We have come to a point where we might start to ask another question: Given the requirements for a system like FL, how well was it designed?

Suppose a super-engineer were given the specifications for the human language system (it must interact with the external systems available, i.e., the human articulatory system, the human interpretation system), and asked to design a device that satisfies the conditions in some optimal way. How close to this kind of optimal design does the human FL come?
The strongest version of the minimalist thesis would be that:

**Language is an optimal solution to legibility conditions.**

This might not be true—in fact, Chomsky takes great pains to point out that it seems highly improbable that there’s all there is to language. Yet, we can advance understanding by seeing what the strong minimalist thesis leads us to, and adding only what “imperfections” are necessary—and if they are truly necessary (and not just apparently necessary). So the program proceeds on the idea that we will see just how far we can get with the strong minimalist thesis.

### 3. Architecture

L defines (recursively) a set of expressions ($\text{EXP} = \langle \text{PF}, \text{LF} \rangle$). One question that has come up in the past is: What kind of system is L? By what means is it constructing this set of expressions? Is it **representational**, a system which says the EXP$_1$ is in the set if it meets some specified set of conditions? Is it **derivational**, a system which says that EXP$_1$ is in the set if you can arrive at EXP$_1$ by running a specified (and recursive) set of procedures? Is there a real difference? For the most part, derivational conditions and representational conditions can be intertranslated, after all...

Chomsky’s current take on this is that there is a real, but very subtle difference, and that the computational system is a **derivational** one. The main evidence for this comes from the appearance that the language system has for minimizing computational complexity, which would be the sort of thing we might expect of a derivational computational system (or at least one which would not be expected to have any relevance for a representational computational system).

Building up the system:

UG makes available a set $F$ of features (linguistic properties) and computational operations $\text{C}_{\text{HL}}$ (the computational procedure for human language) which make use of $F$ to form expressions. A particular language L uses F to form a particular set of expressions EXP.

One step further in, a language L takes those features and organizes them into a lexicon. The lexicon consists of words, each of which has properties (drawn from F). A lexical item can be considered to be a “bundle of features” (something with several linguistic properties, tied together).

We could be convinced that this reduces the complexity of the operations that the language L has to perform; rather than building things up from all of the possible combinations of features, the lexicon provides a limited set of feature combinations from which it must draw in doing a computation.

If convinced, we can take it one step further, and suppose that every computation is done on a pre-selected part of the lexicon, a **lexical array** selected from the lexicon, which will then undergo the computational procedure. Then, the computation need not search the entire lexicon for the elements it is to apply to, it must only choose from this quite limited array.

Note about the lexical array: It is basically the set of words that will be used, except that the notion set isn’t quite right because it is perfectly legitimate to use the same word twice (e.g., *The dog ran after the man*). For this reason it is also called the **numeration**.

Returning to the question of the operations that $\text{C}_{\text{HL}}$ provides. Any language-like system will need at least an operation **Merge**, which takes two syntactic objects and forms a single object from them.

Another operation which we seem to need (as we’ll discuss) is **Agree**, which establishes an agreement relation of some kind between a lexical item and a feature $F$ in some restricted search space. This operation does not seem to be required in any language-like system (e.g., computer programming languages); it seems to be a property of human language, and if the strong minimalist thesis is right, it should follow from the interface conditions.

The last operation is **Move**, which can actually be decomposed into Agree and Merge; Agree establishes a relation between a lexical item (in general, the head to which the moving item will move) and a feature (in general, the feature which designates the moving item as the moving item), and then some syntactic object containing the feature $F$, called $P(F)$, is added to the top of the tree by Merging it with the existing tree.

If Move can really be considered to be a complex operation, consisting of simpler operations Agree, and Merge, and the determination of $P(F)$, then we might be able to be convinced that a well-designed system would avoid the complex operation in favor of the simpler ones if the simpler ones were available. That is, you only use Move if you can’t do what you need to do with just Merge or Agree.
3.2. Fundamental relations: X-bar theory

The computational system takes things from the lexicon and arranges them in structures. We (for the moment) take the relations among them to be limited to those provided by X-bar theory:

\[
\text{(8) } \begin{array}{c}
\text{XP} \\
\text{ZP} \\
\text{X} \\
\text{YP}
\end{array}
\]

There are two local relations represented here: Spec-head, and head-complement (the more local of the two). Another permissible relation is the relation between a head (X) and the head of its complement (Y), involved in selection.

In particular, we will not allow ourselves more complex relations, such as the relation between a head and some part of the complement—which rules out the relation of head-government. Despite the fact that lots of analyses involved head-government, they will all need to be re-analyzed if we wish to pursue the minimalist program.

As an example. Case was often assumed to be assigned under government (by V, for objective Case), although nominative case was assigned by a Spec-head relation to the subject. If we are not allowed government, then we must unify Case assignment to Spec-head relations, which we can do in the spirit of Pollock’s Split-INFL:

\[
\text{(9) } \begin{array}{c}
\text{CP} \\
\text{Spec} \\
\text{C} \\
\text{AgrSP} \\
\text{Spec} \\
\text{AgrS} \\
\text{TP} \\
\text{T} \\
\text{AgrOP} \\
\text{Spec} \\
\text{AgrO} \\
\text{VP}
\end{array}
\]

Now, we say that nominative Case is assigned 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 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 John intelligent, or to SpecAgrSP in John is intelligent).

3.3. Beyond the interface levels: D-structure

D-structure is not an interface level—it does not really belong in the MP.

We have two operations, GT (later: Merge) which takes two objects in the workspace and combines them (in a way which respects X’ theory) into one object; and Move-α, which takes an object a, some phrase within the object itself, and adds it to the object (using the same mechanism as Merge).

There is no more D-structure because by the time we get to the “base positions” of elements in the higher part of the tree, stuff in the lower part of the tree has already moved around.

3.4. Beyond the interface levels: S-structure

S-structure is also not an interface level—can we get rid of it too?

There are many conditions which have previously been assumed to hold at S-structure. In order to say there is no S-structure, we need to show that this was the wrong way to look at those conditions. We need to say that they could apply at LF alone. Better, we could say that sometimes they must apply at LF (not SS), or maybe even that they could not apply at SS.
Binding theory looks like a SS condition:

(10)  a. You said he liked [the pictures that John took]
       b. [how many pictures that John took] did you say he liked it ?
       c. who [t said he liked [α how many pictures that John took]] ?

Idea: Principle C applies to (10); he c-commands John and so he can’t be John.
       In (10a) everything is in its original position (more or less). Principle C applies.
       In (10b), John has moved out as part of the wh-phrase; Pr. C no longer applies.
       In (10c), John is inside an in-situ wh-phrase, assumed to move at LF.
       —The LF representation of (10c) should be a lot like (10b), if all wh-phrases
          have to move to SpecCP by LF. So since Principle C applies to
          (10c) but not (10b), it must be that Principle C does not apply at LF, and
          cares only about SS.
          —But wait—why do we believe that the whole wh-phrase has to move
          to SpecCP at LF? Why not just the how many part (the wh-part). If that’s true, then we would expect (10c) to be ruled out by
          Principle C, even at LF.

So we at least don’t need SS for all Binding Theory cases.