AM2 — Syntax: theories and models
Week 1: theoretical basics

1 Delimitation of the theoretical framework

1.1 A brief summary of the literature

This is a course about the class of theories of syntax that are usually grouped under the labels *transformational grammar* and *Chomskyan syntax*. This class of theories started with the earliest work of Chomsky. The initial stage of transformational grammar encompasses more or less the period going from 1957 until the end of the 1970s. The theoretical framework of this period is characterized by three books (Chomsky 1955, *The logical structure of linguistic theory*; Chomsky 1957, *Syntactic structures*; Chomsky 1965 *Aspects of the theory of syntax*) and three papers (Chomsky 1959, "Review of Skinner's *Verbal Behavior*"; Chomsky 1973, "Remarks on nominalization"; Chomsky 1977, "On wh- movement"). In this class we are going to read parts of *Syntactic structures* and (later on) "On wh- movement".

The second stage, which goes from the late 1970s until the mid-1990s, is defined by the *Government and Binding* framework. The publication that first codified the ideas that dominated this period is Chomsky 1981, *Lectures on Government and Binding*. In 1986, two books followed that expanded the existing framework, namely, *Knowledge of Language* and *Barriers*. In this course, we may or may not read some parts of *Barriers*, depending on time and the topics we discuss.

The final stage, and the one that we will focus on the most, started with the publication of Chomsky 1995, *The minimalist program*. Because of this book's title, this stage is usually called "Minimalism". Chomsky followed up on *The minimalist program* with a series of papers. In chronological order: "Bare phrase structure", "Minimalist Inquiries", "Derivation by phase", "Beyond explanatory adequacy", "Three factors in language design", "Approaching UG from below", "On phases", and the most recent one, "Problem of projection". As many people have noted, these papers get increasingly programatic and decreasingly linguistically-analytic. Therefore, we will mostly concentrate on the theory defined by *The minimalist program*, "Bare Phrase Structure", "Minimalist Inquires" and "Derivation by phase". We will refer to some associated literature (not by Chomsky) to explore some of the ramifications of these proposals.

1.2 Minimalist syntax vis-à-vis other theories

We are going to focus on transformational (minimalist) syntax to the exclusion of other theories of syntax, of which there are possibly many more than you are aware of —for example, *Lexical-functional grammar*, *Head-driven phrase structure grammar*, *Relational grammar*, *Functional grammar*, *Role and reference grammar*, *Word grammar*, *Categorial grammar*. Each of these comes with a number of variants where a few basic assumptions are different (for example, Head-driven phrase structure grammar evolved from *Generalized phrase structure grammar*, which itself was developed in parallel to *Korean phrase structure grammar*). To understand what kind

of theory we are going to be talking about in this course, we need to list its defining characteristics.

Minimalist syntax is (mostly) rule-based A *rule* is a mapping function:[1] it takes a specific structure (what used to be called a *structural description*) and returns a different, but closely related structure. This contrasts with *constraints*, which are penalty functions: they take a structure as their input, but their output is not another structure. Rather, it is a natural number, which is zero if the structure satisfies a series of conditions, and something larger than zero if it violates some of those conditions.

Minimalist syntax is mostly rule-based. We create a chain of rules $r_1, \ldots, r_n$, with the input of r_i being the output of r_{i-1}, for any $i > 1$. If everything goes well, the output of the last rule is a well-formed sentence. If this happens, we say that the derivation *converges*. Occasionally, we will encounter a constraint (which, for historical reasons, is occasionally called a *filter*), which will output a grammaticality violation, rather than another structure. In this case, we say that the derivation *crashes*. For example, Binding Conditions are commonly stated in terms of constraints, rather than rules. There have been some efforts to make minimalist syntax *entirely* rule-based, in what is occasionally called *crash-proof syntax*. For the most part, though, I will not say a lot about these efforts.

Minimalist syntax is proof-theoretic Usually, people use the term *generative* instead of *proof-theoretic*, but I feel that *proof-theoretic* is more useful when it comes to understanding this property. Syntax, as envisioned by Chomsky and his followers, is a mathematical system in the formal sense of the word. One has a set of axioms and a set of rules of derivation, and by applying the rules to the axioms, individual sentences follow as theorems within the system. In informal talk, we say that we *derive* or *generate* a sentence, but in reality, theorems aren't derived or generated —they are *proven*, hence the term *proof-theoretic*.

In opposition to proof-theoretic syntax we have *model-theoretic* syntax, mostly advocated by Postal in his Relational Grammar framework. The difference between proof- and model-theoretic syntax reflects a disagreement between Chomsky and Postal on what the object of study of linguistics (and specifically, syntax) ought to be. Chomsky argues that linguistics ought to investigate the *knowledge* that humans have of their respective natural languages. In contrast, Postal argues that the object of study must be natural language as a disembodied entity (note that this presupposes that language and knowledge of language are different things). In a model-theoretic syntax, one begins with an infinite set of sentences. These sentences don't come from anywhere, they are just assumed to exist. Then, one defines a series of constraints that delimit a proper subset of sentences, namely, those that are found grammatical by the speakers of the relevant language. Since the sentences exist as such from the beginning, no proof (derivation) of their well-formedness is necessary; one just needs to check whether they satisfy the constraints.

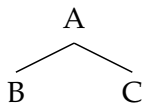
Minimalist syntax is strictly endocentric This property simply means that every constituent must have a head. A head, in turn, is a constituent from which a phrase inherits certain features (mostly, categorial features). Formally, we say that if we combine ("merge") two constituents with labels α and β, then the label of the resulting superconstituent is going to be either α or β. For example, if we merge a verb (category label V) and a nominal phrase (category label DP), then the resulting constituent will have the verb's label, and we will call it a VP.

[1] Logic/math reminder: a *function* always gives the same output for any given input. If we can get a variety of outputs for a single input, then we don't have a function, we have a *relation*. So, a function is a specific type of relation, namely, a *one-to-one* relation.

Note that this is not a feature in many other frameworks, or even in previous stages of transformational grammar. There, some constituents do not have a head, in the sense outlined above. For example, all the way up to *Barriers*, the combination of a VP and a subject DP created a constituent with category label S (sentence). In the current framework, we call this constituent a TP, which reflects the idea that there is an underlying head T mediating between the subject and VP.

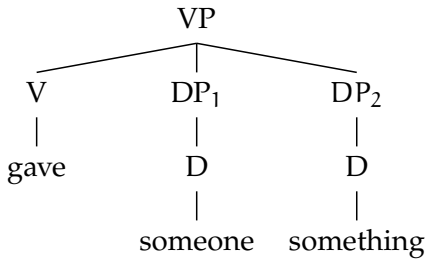
Minimalist syntax is strictly binary branching In current minimalism, every non-terminal node is going to immediately dominate *exactly* two other nodes. In other words, every proper subpart of a syntactic structure is going to look like this:

(1)



Again, this is something that is not true of other frameworks or even previous stages of transformational grammar, where both less-than-binary (unary) and more-than-binary (ternary, quaternary,...) branching is allowed. For example, it used to be common to analyze double object constructions by means of a combination of ternary and unary branching.

(2)



As we will see in this course, the switch to a strictly binary branching structure is empirically motivated (although it was originally proposed for purely theory internal reasons, *viz.*, Kayne's 1984 notion of *connectedness*). The motivation is that there are data that point to an asymmetric structure in all the relevant cases —i.e., in double object construction, the first object is higher up than the second, not at the same level.

Minimalist syntax is bottom-up derivational As mentioned above, in minimalist syntax we construct sentences by combining lexical items and small constituents into larger constituents. Crucially, the standard way of doing this is by starting at the bottom of the structure (the most embedded constituents) and proceeding upwards. This differs from the assumptions in the earliest stages of transformational grammar, where one started with the root node (notated Σ) and then proceeded downwards. The rules that defined this kind of grammar were called *rewrite* rules, and they looked like this.

(3) a. $\Sigma \rightarrow S$.
b. $S \rightarrow NPVP$.
c. $NP \rightarrow DetN$.
d. $VP \rightarrow VNP$.
e. ...

Within the context of minimalist syntax, some people (most notably Colin Phillips and the *Dynamic Syntax* group at King's College London) have defined alternatives in which structures are built top-down, as they were with rewrite rules. One crucial difference between the two ways of building a structure is that bottom-up derivations are structure preserving —i.e.,

if α and β form a constituent at a certain derivational step, they will also form a constituent at any subsequent derivational step. In top-down derivations, this is not the case: α and β may form a constituent at derivational step n, but the addition of γ at step $n+1$ might break up this constituency.

2 The four pillars of minimalist syntax

Now that we know what we want our syntax to look like, we can start discussing its building blocks. We are going to see that minimalist syntax (actually, arguably any theory of syntax) must have four components in order to be usable: (i) features; (ii) lexical items; (iii) a computational system; and (iv) interfaces.

2.1 Features

A feature is an undivisible bit of information about syntax. Note the word *undivisible*: if a purported feature can be divided into smaller features, then we don't have a feature, we have a *feature matrix* or a *feature set*. The three different types of features (which are not entirely compatible, so it's important to know which one a specific author is assuming) are (i) multivalued; (ii) binary; and (iii) privative.

Multivalued features These features have a bipartite internal structure, consisting of an *attribute* and a *value*. The attribute is essentially a label for a set of pieces of grammatical information that constitute alternatives to each other. For example, [masculine], [feminine], and [neuter] form the attribute gender. The value is the instantiation of an attribute on each individual case. Here is an example of a feature. By convention, we enclose the features in square brackets ([ ]) and use a colon (:) as a metagrammatical symbol to separate attributes from values.

(4) [gender : masculine]

As their name indicates, multivalued features can choose their value out of a set of arbitrary size. Note that, while an attribute can take any one value out of a given set of values, it may only take a single value at a time. Thus, the following is not a well-formed feature.

(5) * $\left[\text{gender} : \begin{matrix}\text{masculine}\\ \text{feminine}\end{matrix}\right]$

Binary features In a binary feature system, we retain the [attribute : value] distinction. However, the values of multivalued features are reclassified as attributes, and then we take the symbols, + or − as the only possible values for these new attributes. The name *binary features*, obviously, comes from the fact that these features only have two possible values. In this system, (4) can be rewritten as follows.

(6) [masculine : +]

In contrast, non-masculine gender would be written as follows.

(7) [masculine : −]

Privative features In a privative feature system, we eliminate the [attribute : value] distinction entirely. By writing an attribute, we directly entail a + value. Feature (6) can be rewritten privatively as (8).

(8) [masculine]

In a privative system, negatively-valued features don't exist. Whereas in a binary system [masculine : −] is the opposite of [masculine : +], in a privative system there is no opposite for [masculine]. There is simply a [masculine] feature, or there is none.

2.2 The lexicon

To define what the lexicon is, first we need to define the notion of *lexical item*. As you might have imagined, we can put features together to form larger feature matrices. For example, the pronoun *er* consists of the following matrix (expressed using multivalued features).

(9)
$$\begin{bmatrix} [\ \text{category} : \text{D}\ ] \\ [\ \text{number} : 3\ ] \\ [\ \text{person} : \text{singular}\ ] \\ [\ \text{gender} : \text{masculine}\ ] \\ [\ \text{case} : \text{nominative}\ ] \\ [\ \ldots : \ldots\ ] \end{bmatrix}$$

We want to say that *er* is a lexical item, but something like *der Mann* is not, despite having very similar features. To capture this difference, we are going to assume the following definition.

(10) *Lexical item*
A lexical item is a *built-in* (pre-assembled) feature matrix.

The lexicon, therefore, is the component that contains all lexical items, *qua* pre-assembled feature matrices. We can then put lexical items together to build larger phrases like *der Mann*. Occasionally, things that look like phrases can count as lexical items. An obvious case is idioms, where the meaning of a phrase is conventionalized, rather than compositionally derived.

2.3 The computational system

This is where we are going to spend most of our time until the Christmas break. The computational system is what people have in mind when the word "syntax" is mentioned. For us, it is going to contain two things:

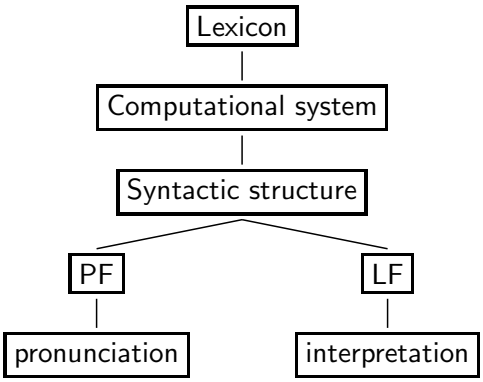
1. the rules (mapping functions) that allow us to construct large phrases out of lexical items and smaller phrases.
2. some constraints (penalty functions) that will allow us to exclude certain classes of ungrammatical structures.

We are not going to worry too much about constraints, since they are not central to minimalist syntax. With respect to rules, the earliest writings on transformational grammar had literally hundreds of rules. However, these were highly specific, and the trend ever since has been to reduce their number by making them more general. Currently, we are left with just two core rules, namely, *Merge* and *Agree*. For the time being, you only need to know that they exist; we will worry about the details in the next few classes. However, here is a quick summary of what they entail.

- *Merge*: take two syntactic objects α and β (either phrases, lexical items, or a combination) as an input. The output is a phrase γ which has either α or β as its head. This subsumes movement operations, which are redefined as *internal* Merge.
- *Agree:* take as input two features, the *Probe* (unvalued) and the *Goal* (valued), such that the lexical item containing the Probe c-commands the lexical item containing the Goal. In the output, the value of the Goal has been copied to the Probe.

2.4 The interfaces

So far, the combination of features, lexical items, and the computational system has given us all we need to construct syntactic structures. However, syntactic structures alone are pretty useless. In order to be useful, they have to be *pronounced* and *interpreted*. The component that allows syntax to communicate with phonology and prosody is called *Phonetic Form* (PF); conversely, the component that allows syntax to communicate with semantics is called *Logical Form* (LF). Standardly, they are assumed to take the output of syntax as their inputs, which gives rise to the *Y-model*:



Note that, under this architecture, syntax communicates with PF and LF, but PF and LF don't communicate with each other. This might get us into trouble when we look at *focus* and *ellipsis* phenomena, in which a certain interpretation is dependent on a certain pronunciation. An example of a focus phenomenon is question-answer congruence (here, capitals indicate main sentential stress; $[_F\ ]$ indicates the semantic focus of the sentence). Generally, the main sentence accent has to be contained inside the $[_F\ ]$-marked constituent.

(11) A: What did Alice eat?
B: ✓ She ate $[_F$ a COOkie].

(12) A: What did Alice eat?
B: * SHE ate $[_F$ a cookie].

We will look at interface phenomena in Block 3 of this course, so we don't have to worry about these things until after the Christmas break.

3 Reading for October 18

Chomsky, Noam. 1957. *Syntactic structures*. The Hague: Mouton (read chapters 2–4).