Lecture 16

Derivations as the Primary Data Structure

Our formal understanding of syntax has matured quite a bit over the last few weeks, but it seems that we have reached an impasse. On the one hand, the hypothesis that syntax is definable in first-order logic seems to be on the right track when it comes to expressing syntax dependencies: virtually all constraints in the syntactic literature fit into this class, and where constraints in their canonical formulation would require more powerful computational machinery — as is the case with NPI-licensing and binding theory — a closer look at the data actually supports a more restricted formulation that does not exceed the limits of first-order logic. When formal predictions line up with empirical data so nicely, the odds are good that one has identified a relevant property.

On the other hand, there are weak-generative capacity arguments that first-order logic is not enough. Since the first-order definable tree languages are just another dot in the hierarchy of tree languages between CFGs an refined strictly local, a first-order definable set of phrase structure trees can only generate context-free string languages. But constructions like reduplication in Yoruba (Kobele 2006) produce string languages of the form a^{2^n}, which is not context-free. As you might recall, weak generative capacity arguments are the strongest argument one can make for establishing a lower bound; at the very least, a grammar formalism must be able to generate the correct string language. So unless we want to doubt the Yoruba data (and several other constructions in completely unrelated languages), it seems that we have unassailable proof that syntax is not first-order definable.

Remember that the fact that a language contains a proper subset of complexity c does not imply that the whole language has at least complexity c. After all, the regular string language $(aa)^+$ is a subset of the strictly 0-local Σ^*. But Kobele (2006) uses the right proof technique to show that Yoruba as a whole is not context-free.

Today we will see that this conclusion is not quite right, and that a more refined view can accommodate both the relative weakness of syntactic constraints over trees and the much greater expressivity over strings. This approach will also solve a variety of other problems, such as how to accommodate copy movement as well as string patterns that do not involve copying yet are nonetheless not context-free.

1 The Role of Syntax

1.1 Structural Descriptions

In our discussion so far, we have treated syntax as a mechanism that generates tree languages. Our original motivation for this step was that tree languages make it very easy to generate non-regular string languages. We could have used a different strategy.

For instance, a finite-state automaton can be turned into a *push-down automaton* (PDA) by adding a stack that can memorize an unbounded number of symbols but may only read or manipulate the top-most symbol of the stack (see e.g. Sipser 2005). The transition rules of a PDA are expanded FSA transitions that also take the top symbol in the stack into account. One can show that the stack symbols correlate in an abstract sense with the non-terminal nodes of a tree, but that does not mean that a PDA generates tree languages of any kind. Its output is string languages, and the class of string languages recognized by a PDA is exactly the class of context-free languages. So trees were not an inevitable choice for our treatment of syntax, instead of enriching our data structures from strings to trees we could have adopted a more sophisticated memory architecture.

But we opted for trees because they line up with how linguists think of trees. There is plenty of evidence that sentences have an internal tree-like structure. Some arguments pertain to how children generalize from input when acquiring their language, some hinge on prosody, others on semantic scope. Syntactic constraints also seem to pay attention to structural factors rather than just string properties. And formally, of course, the usage of trees revealed a nice parallel between phonology and syntax such that the technical machinery we employed for the former could readily be applied to the latter. So a tree-based view of syntax is ultimately a lot more appealing.

Syntax, then, is a mechanism for generating tree languages such that each tree is a structural description of some sentence. Every linguist's natural inclination at this point would be to equate structural description with phrase structure trees. But nothing what we have said so far supports this conclusion. One can imagine many different kinds of tree-like structural descriptions. A dependency tree, for instance, encodes the head-argument relation rather than phrase structure, which must be deduced from the structure (whereas the head-argument relation must be deduced in a phrase structure tree with movement). An example of the two tree structures is given in Fig. 16.1. The phrase structure tree dwarfs the dependency tree in size, yet one can freely convert between the two formats given a fixed theory of what phrase structure trees look like.

The phrase structure tree ties the grammatical relations like subject, direct object and indirect object to specific positions in the tree and then displaces various subtree in order to obtain the desired word order. The labels and projections only matter to the extent that they identify the relevant positions and regulate what may be displaced. The dependency tree explicitly represents the grammatical relations and completely ignores word order, which must be computed through a separate mechanism, e.g. a mapping from dependency trees to derivation trees. Given this kind of malleability, it is a lot less clear what kind of structural description syntax operates over.

1.2 Interfaces

Another argument for phrase structure trees is that they (supposedly) serve as the input to what's called the *interfaces*, PF and LF. LF stands for *logical form* and is the interface to semantics, i.e. where the meaning of sentences is computed. PF stands for *phonetic/phonological/physical form* and covers all kinds of aspects related to uttering the tree, e.g. computing its string yield. The interplay between syntax, PF and LF is usually depicted via the (inverted) T-model.

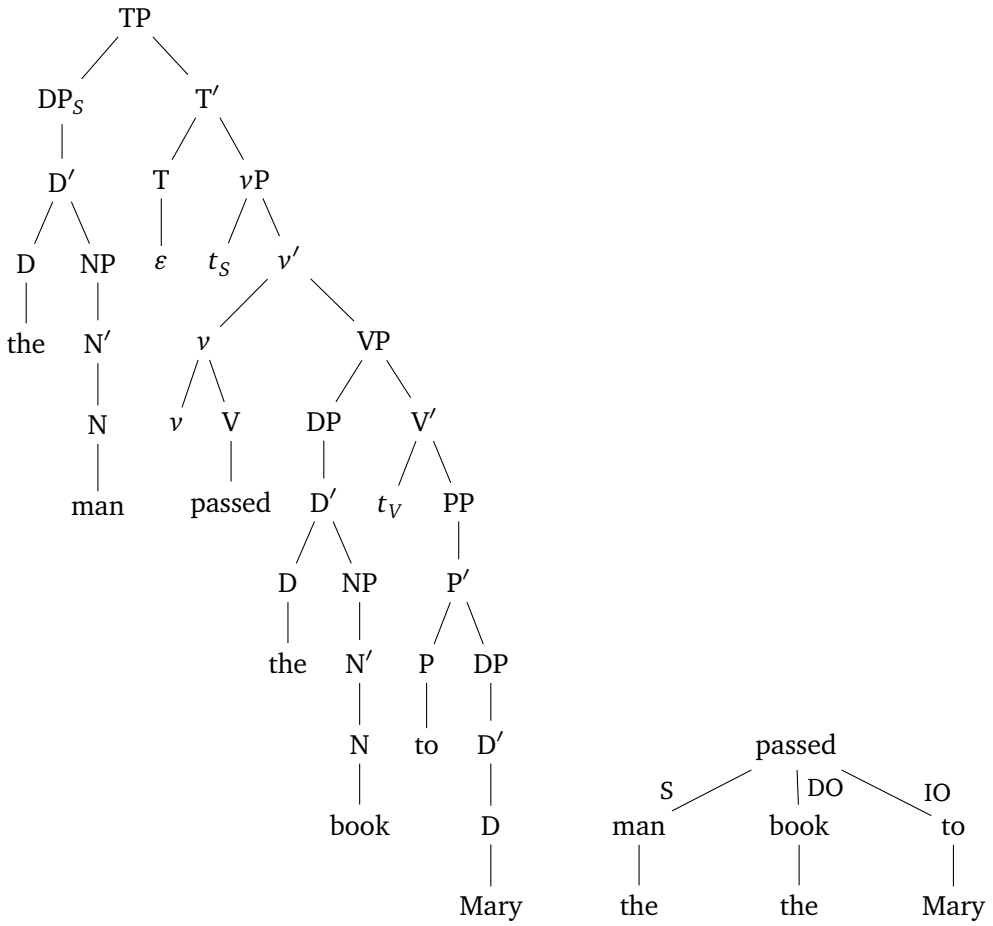
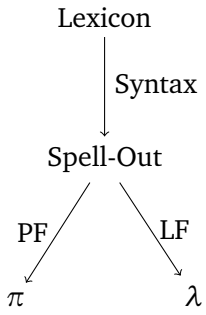


Figure 16.1: Phrase structure tree and dependency tree for *The man passed the book to Mary*.



The idea is that syntax combines a collection of lexical items into a phrase structure tree. Once this assembly process is finished, the operation Spell-Out passes on the tree to PF and LF, which apply further modifications to yield an output string π and some kind of logical formula λ.

Before we determine whether the T-model favors a particular kind of tree structure, we should establish first that this is a reasonable model of language. After all, the T-model has been extensively criticized by various linguists on the grounds that it, supposedly, puts too much emphasis on syntax and rules out any kind of PF and LF interaction. A common complain is also how syntax can pick the right lexical items and build the desired tree for a given meaning. This criticism is confused and stems from a common mistake among linguists: criticizing a formalism for the intuition that is used to make it more approachable. The T-model has nothing to say about the actual building process, it is not a model of generation. What is describes is a collection of languages and the functional relations between them. The lexicon is a finite collection of lexical items, and the set T of syntactically well-formed phrase structure trees is the image of the lexicon under a function we call syntax. Similarly, the sets of strings and meanings of the language, respectively are π, the image of T under the PF mapping, and λ, the image of T under the LF mapping. All the T-model establishes is sets and mappings between these sets.

This also means that arguments about derivational versus representational syntax, and one-level versus multiple levels are all rather misguided. They might aid the intuitions of researchers and lead them to posit interesting conjectures or seek out new data, but on a mathematical level there is no sense in which mappings are clearly derivational or representational.

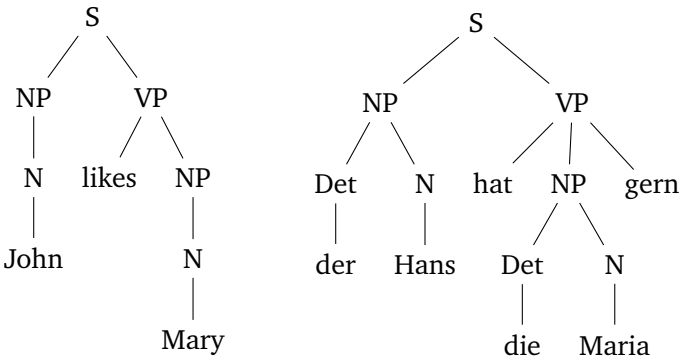
Mappings have no inherent directionality, for given a relation R one can always take its inverse R^{-1}. So producing a string to express meaning ϕ is tantamount to computing $\text{PF}(\text{LF}^{-1}(\phi))$. Figuring out the meaning of string s, on the other hand, means computing $\text{LF}(\text{PF}^{-1}(s))$. All the T-model does, then, is to establish syntax as a mediator between these two kinds of objects, physical utterances and logical formulas: the process of mapping one to the other is analyzed as first constructing a fitting tree structure which is then fed into the correct mapping. Yet at no point does syntax "come first", and the arrows in the T-model only indicate the direction in which the individual mappings are defined, not the direction in which the mapping is run.

The T-model does not even make a bold ontological claim that one could take issue with. Whenever one has a relation R between two sets S and T, this relation can be decomposed into a *bimorphism*, which consists of two mappings R_1 and R_2 and an *interpolant* L such that S is the image of L under R_1, T is the image of L under R_2, and $R = R_1^{-1} \circ R_2$ (Arnold and Dauchet 1982). That is exactly the scenario described by the T-model: S is the set of output strings, T the set of logical formulas, L is the set of structural descriptions, R_1 is PF, and R_2 is LF. If one takes syntax to be a finite description of the interpolant L based on a language's lexicon, the place of syntax in the T-model becomes an unavoidable mathematical truth (and any interaction between PF and LF happens in syntax by definition).

The flip side is that syntax may be just a mathematical truth, with only π and λ enjoying any degree of cognitive reality. That would not be particularly shocking, though, since one of the main advantages of abstraction is that objects can be posited irrespective of whether they are real — monoids do not exist in any verifiable sense, but they are a useful abstraction and unification of a variety of distinct objects. Even if syntax isn't real, it plays an important part in modeling the behavior of a real cognitive ability.

Another point supporting the generality of the T-model is that the very same perspective is becoming increasingly more important in machine translation. In machine translation, one has to translate freely between two languages, e.g. English and Chinese. The longest time, tree-based approaches took the form of *synchronous grammars*, which generate two trees in parallel. For example, the following synchronous CFG translates *John likes Mary* into Bavarian German *Der Hans hat die Maria gern*, and the other way round.

$$
\begin{array}{rcl}
\text{S} & \rightarrow & \langle \text{NP VP}, \text{NP VP} \rangle \\
\text{NP} & \rightarrow & \langle \text{N}, \text{Det N} \rangle \\
\text{Det} & \rightarrow & \langle , \text{der} \rangle \mid \langle , \text{die} \rangle \\
\text{N} & \rightarrow & \langle \text{John}, \text{Hans} \rangle \mid \langle \text{Mary}, \text{Maria} \rangle \\
\text{VP} & \rightarrow & \langle \text{likes NP}, \text{hat NP gern} \rangle
\end{array}
$$



Somewhat surprisingly, synchronous grammars are not particularly well-behaved, with minor variants yielding vastly different formal properties. But this variability can be studied more insightfully from the bimorphism perspective. In our example, one would posit an interpolant that represents the structural similarities between the two trees, and two mappings that turn the interpolant into the English and German tree, respectively. The formal differences between variants of synchronous CFGs are then revealed to give rise to mappings of greatly differing complexity. These differences are very hard to state as properties of the direct mapping between languages, but are highly transparent from the bimorphism perspective (see e.g. Shieber 2004).

Overall, the T-model represents a very general idea of decomposing mappings. In fact, since the basic task of any native speaker is to translate sentences into thoughts and the other way round, it isn't too surprising that the T-model looks exactly like a modern machine translation template. The big question at this point is whether this tells us anything about what the interpolant should look like. In other words, what sort of information needs to be present in the structural descriptions furnished by syntax?

2 Interpolation via Derivation Trees

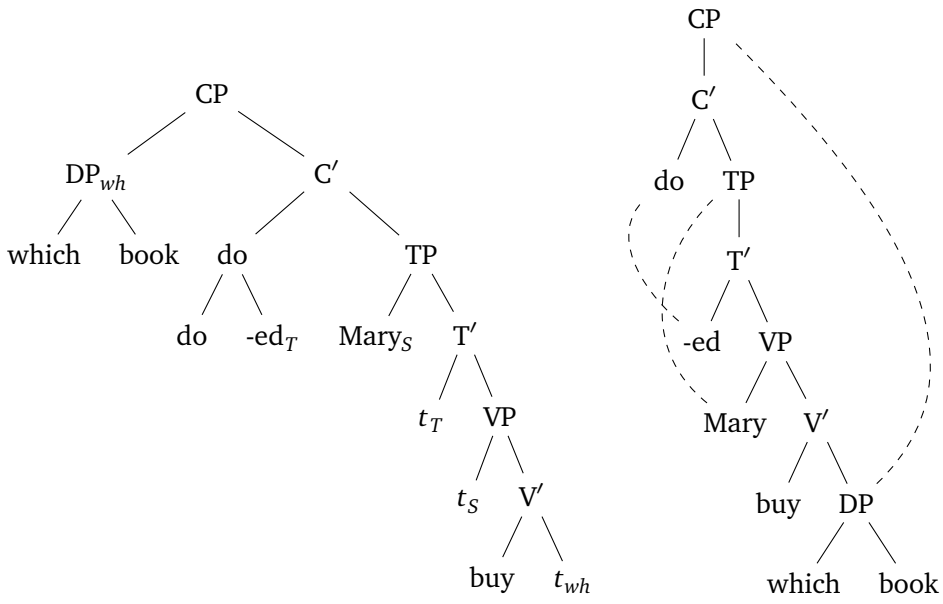
2.1 Movement without Displacement

We are still committed to the idea that syntax is first-order definable, but we have now refined our conjecture so that it applies to the structural descriptions that are regulated by syntax. These are not necessarily phrase structure tree, but some interpolant that

provides all the necessary information so that PF and LF can produce the desired outputs π and λ. Seeing how phrase structure trees are still used in semantics and research on prosody, we do not want to stray too far to them. If a minor variant of phrase structure trees is enough to give us a first-order definable interpolant, this is sufficient support for our conjecture.

What exactly is it, then, that makes phrase structure trees the most commonly used data structure in linguistics? One advantage is that they still provide an implicit encoding of argument structure, even if it is partially obscured by displacement. Argument structure is obviously essential for determining basic thematic relations in a sentence, e.g. who is punching whom. Phrase structure trees also encode semantic scope, e.g. that the universal has scope over the existential in *Every student that took some history course wound up bored*. But the LF mapping is not entirely straight-forward since semanticists often undo displacements or add new instances thereof in order to get the correct scope. This contrasts with the PF mapping, which only has to read the leafs from left to right in order to get the output sentence (we ignore prosody and other phonological factors here for the sake of exposition). Right now, then, the phrase structure trees fully encode movement to keep PF simple, even though it leads to complications on the LF side. There is no *a priori* reason why the workload should be distributed this way. We could just as well leave displacement to the PF mapping so that LF no longer has to undo movement steps — and as we will see next, that is exactly the answer to our problem.

Suppose that instead of moving phrases in our syntactic trees, we simply indicate what is supposed to move where, e.g. via a dashed branch. That way, LF still gets all the information it needs to compute relative scope, while PF can carry out the actual displacement. Compare the two trees for *which book did Mary buy* below (we do not use an X′ template here in order to keep the trees as simple as possible).



Obviously PF can convert the tree to right into the one to the left by making each mover the daughter of the highest node it is related to via a dashed branch. In fact, we can even define this mapping in first-order logic.

2.2 The PF Mapping

So far we have only used logic to define constraints, but it is actually fairly easy to write logical formulas that define mappings between trees. The idea is that we can write formulas that express instructions of the form "if this relation holds between nodes x and y in the input tree, then the following relation must hold between them in the output tree".

Example 16.1 A First-Order Definable PF Mapping

Suppose that we already have a predicate $\text{occ}(x, y)$ that holds between x and y iff x is a landing site (= occurrence) for y, i.e. x immediately dominates y via a dashed branch. Then we first define a predicate to pick out the final occurrence of y, which is where it will be pronounced in the string. Due to how movement works, the final occurrence of y will always be the highest node occurrence of y.

$$\text{f-occ}(x, y) \iff \text{occ}(x, y) \wedge \neg\exists z[z \lhd^{+} x \wedge \text{occ}(z, y)]$$

Now we define a predicate $\blacktriangleleft$ that represents immediate dominance in the output tree. We specify that $x \blacktriangleleft y$ iff either x is the mother of y and y does not move, or x is the final landing site of y.

$$x \blacktriangleleft y \iff (x \lhd y \wedge \neg\exists z[\text{occ}(z, y)]) \vee \text{f-occ}(x, y)$$

A similar strategy can be used to define the left-sibling predicate such that movers are the left sibling of the daughter of their final occurrence and all other nodes inherit the sibling specification from the input tree (if we assume that syntactic trees are unordered, this information must be inferred from some other relation like c-command). With the left-sibling relation, it is very easy to compute the string yield of the output tree, which is exactly the utterance represented by the input tree.

The current PF mapping gives us non-copying movement, but it is very simple to add in copy movement. In fact, copy movement is much easier to handle because it corresponds to a graph-theoretic operation known as *unfolding*, which turns a graph into a tree: if a node n has i mothers $m_1, \ldots, m_i$, make i copies of the subtree rooted in n and make each copy a daughter of exactly one of n's mothers. Unfoldings are not definable in first-order logic, but we can combine them with the previous transduction. First, we assume that each mover has a feature that tells us whether is undergoing normal movement or copy movement. Then we refine $\blacktriangleleft$ so that normal movers are still attached to only their final occurrence, whereas copy movers are attached to all their occurrences. We then compose this mapping with an unfolding to get a PF mapping that can handle both copying and non-copying movement.

Notice that the existence of copying movement is no longer an obvious threat to our conjecture that syntax is first-order definable. While the PF output structures may not fit into this class, the structural descriptions of syntax might since they do not contain multiple copies of a tree. But it might still be the case that these kind of augmented trees with dashed branches are not first-order definable for some other reason. As we will see next, that is not the case either given an empirically supported constraint on displacement.

2.3 Doing Away With Movement Branches

The question whether augmented trees with dashed movement branches are first-order definable intersects with another issue, which is whether the branches are needed at all. Remember that we want syntax to still use tree structures since those are well-understood objects and can automatically be converted into CFGs, which makes them amenable to a variety of standard techniques (e.g. from the parsing literature). So we want to get rid of the branches without losing track of what is moving where.

Let us assume for the sake of exposition that every phrase moves at most once and that every phrase is a target for at most one mover. So TP, for instance, can only be targeted by the subject, and the subject moves directly to TP without going any farther or stopping somewhere else on the way. We can represent this fact by adding a feature TARGET with value *nom* to the AVM of TP's head, and the corresponding DISPLACE with value *nom* to the head of the subject. As long as there is no other head with a *nom*-valued feature, the features make it clear that the subject moves to SpecTP, so we do not need a dashed branch to indicate this information — it can be inferred from the feature calculus.

Example 16.2 A Feature-Based Definition of Occurrences

The feature-based specification of movement allows us to define the predicate $\text{occ}(x, y)$ from the previous example without making reference to dashed branches. Instead, the occurrence of a phrase (remember that we assume every phrase moves at most once, so it cannot have more than one occurrence) is the closest dominating node whose head has a target feature matching the displace feature of the head of moving phrase. To make the formula we readable, it is preferable to first define the notion of a potential occurrence via feature matching and then define occurrence as the closest potential occurrence.

$$\text{p-occ}(x, y) \iff \exists t \exists d[\text{head}(t, x) \wedge \text{head}(d, y) \wedge \bigvee_{l \text{ has target feature } f} l(t) \wedge \bigvee_{l \text{ has displace feature } f} l(d)]$$

$$\text{occ}(x, y) \iff \text{p-occ}(x, y) \wedge \neg \exists z[\text{p-occ}(z, y) \wedge x \lhd^{+} z]$$

The assumption that every feature occurs on at most one head in a tree is highly unrealistic. For example, a sentence with multiple embeddings, e.g. *John said that Mary believes that Peter likes her*, involves multiple subjects that all move to their respective SpecTP, and we do not want to posit different features for these different instances of subject movement. Fortunately this isn't necessary. In the example above, we defined *occurrence* — our logical analogue to dashed branches — as the closest node with matching target feature. In a sentence with multiple embeddings, the closest potential occurrence for each subject is the TP in the same clause. So we can allow multiple heads to have the same displace feature as long as they each have distinct occurrences. We stipulate that all trees in which this is not the case are ungrammatical, which can even be enforced via a first-order constraint:

Determinism $\forall x \exists t \Big[\Big(\text{head}(x, t) \wedge \bigvee_{l \text{ has target feature } f} l(t) \Big) \rightarrow \exists! y[\text{occ}(x, y)] \Big]$

At first sight determinism does not look empirically tenable since there are plenty of cases where two phrases move to the same target due to the same feature, e.g.

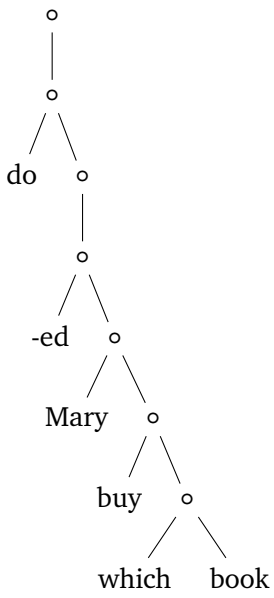
in multiple wh-movement. But I) these cases have competing analysis where the wh-phrases form a cluster that moves once, rather than all phrases moving at the same time, and II) it is unclear whether multiple wh-movement is unbounded. As long as there is a finite bound on the number of parallel movers, the data can be brought in line with Determinism. The crude but simple option is to distinguish multiple features of the same type, e.g. wh_1, wh_2, ... instead of just wh. A more elegant solution is to provide a first-order definable locality mechanism that decides the relative movement order between phrases, thus preserving the intuition of Determinism that we can correctly infer dashed branches just from feature specifications.

2.4 Derivation Trees and Minimalist Grammars

We now have an incredibly flexible and powerful yet first-order definable theory of syntax.

1. Lexical items are richly annotated AVMs whose features determine subcategorization and movement dependencies.
2. Syntactic trees are label-free and represent movement only indirectly through the features.
3. Movement is regulated by the first-order definable occurrence predicate and the first-order constraint Determinism.
4. The PF mapping turns syntactic trees into phrase structure trees. The non-copying part of the mapping is first-order definable.

Notice that since trees are label-free, our tree for *which book did Mary buy* should actually look like the one below (with AVMs omitted).



Unary branching nodes indicate where movement takes place. In the jargon of the dominant theory of syntax, *Minimalism*, we can identify them with instances of the Move operation. The binary branching nodes on the other hand correspond to items being combined via subcategorization, which is handled by Merge. So the structural

descriptions that we have converged on due to our desire to maintain first-order definability are the derivation trees of the leading syntactic framework. In a certain sense we have come full circle, seeing how originally we viewed phrase structure trees as the derivation trees of CFGs.

The moral of the story is that even though CFGs are not powerful enough to generate the right string languages for syntax, that shortcoming is due to their simple PF mapping where the string yield is obtained by simply reading the leaves from left to right. In our new system, we still have very simple tree structures, but the PF mapping is more complex since it now also has to take movement into account. But once the complexity of computing the string yield is factored out, the remainder is a first-order definable tree language that can accommodate all syntax dependencies.

This is the core insight of recent work on *Minimalist grammars* (MGs), a formalization of Minimalism by Stabler (1997). In the last 10 years, MGs have generalized the ideas above to allow for new movement types, provide a corresponding LF mapping, and much more. For an extensive overview, see Graf (2013).