Our linearization theory involves a procedure that collects sets of ordered pairs, and a couple of filters that interpret those ordered pairs as precedence relations. The procedure that collects ordered pairs is Kayne’s LCA, slightly modified.

1. \( \alpha \) c-commands \( \beta \) iff every phrase that dominates \( \alpha \) dominates \( \beta \), and \( \alpha \) does not include \( \beta \).
2. \( \alpha \) asymmetrically c-commands \( \beta \) iff \( \alpha \) c-commands \( \beta \), and \( \beta \) does not c-command \( \alpha \).
3. \( d(X) = \text{def.} \) the set of all terminals dominated by \( X \).
4. \( d((X, Y)) = \text{def.} \) the set of all ordered pairs \( (x, y) \) such that \( x \) is dominated by \( X \) and \( y \) is dominated by \( Y \).
5. \( d((X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)) = \text{def.} \) \( d((X_1, Y_1)) \cup d((X_2, Y_2)) \cup \ldots \cup d((X_n, Y_n)) \).
6. Let \( A \) be a set \( (X, Y) \) from some phrase marker such that \( X \) asymmetrically c-commands \( Y \), and \( X \) and \( Y \) are either maximal or minimal projections.
7. \( d(A) \) is a linearization.

This procedure is subject to constraints that ensure that the resulting set of pairs is a well-formed ordering:

(2) Constraints on a Linearization Algorithm

1. If \( x < y \) and \( y < z \), then \( x < z \). (transitive).
2. For all distinct terminals \( x \) and \( y \) in a phrase marker, either \( x < y \) or \( y < x \). (total).
3. not \( (x < y \land y < x) \) (antisymmetry).

Note that, unlike in Kayne’s system, \( A \) isn’t required to contain all of the asymmetric c-commanding pairs in a phrase marker. It can be any subset of those pairs. That means that \( A \), in principle, collect so few asymmetric c-commanding pairs that \( d(A) \) will not be a total ordering. We’re using Totality to rule out those cases. Totality, then, requires that \( A \) contain enough asymmetric c-commanding pairs that all terminals end up in the ordering. This move has a consequence for our theory of movement that I will explore in this lecture.

I’ve also suggested that we could capture Word Order typology differently than does Kayne. Kayne suggests that the ordered pairs in \( d(A) \) are all interpreted as precedence:

\[ \langle x, y \rangle = x < y \]

This allows only SVO type languages. The others, he suggests, are produced putting constraints on s-structures that require movement to change the relative order of heads and the phrases their projections contain. SOV word orders, for instance, are produced by forcing leftward movement of the complement of a V to a position before it.

The alternative I sketched would let the orderings that \( d(A) \) forms be interpreted in different ways depending on whether phrasal movement is involved or not. In addition to (2), I suggested that we add what the former headedness parameters say. I did that with something that resembles alignment constraints:

(4) a. **Leftwards**
   
   If \( \alpha \) moves and merges to \( \beta \), then the left edges of \( \beta \) and \( \alpha \) must align.
   
   b. **Headedness**
   
   If \( \alpha \) immediately contains \( \beta \), and \( \alpha \) is not a projection of \( \beta \), then the left, or right, edges of \( \beta \) and \( \alpha \) must align.

These are informal constraints, as they make reference to “edges,” which are objects not straightforwardly available in a formalism that conceives of linearizations as orderings rather than strings. I’ll attempt to do a better job of expressing these constraints today.
Our theory of movement involves four processes.

1. Make a copy of an X^0 or XP.
2. Put that copy into the position specified by the relevant rule.
3. Optionally apply trace conversion to the lower (copied) phrase if that phrase is a DP.
4. Remove either the copy or the original term from the linearization.

Our original formulation of movement rules said things like this:

1. Move an XP into the Specifier of YP.
2. Right Adjoin an XP to the smallest IP dominating it.
3. Adjoin an X^0 to Y^0.

We don't have the notion of a Specifier any longer. Instead, we can express

1. Left adjoin the copy to YP.

We've adopted the hypothesis that phrasal movement is always leftwards, which means we can reduce this to:

1. Adjoin the copy to YP.

"Adjoin" is the special instance of merge that involves phrases. So we can reduce (6a) and (6c) to:

1. Merge the copy to µ.

We lose Heavy NP Shift on this view, so we'll have to recapture those word order effects, perhaps in the way that Larson and Kayne suggest. Our theory of movement now is:

1. Make a copy of an X^0 or XP,
2. Merge that copy to µ,
3. Optionally apply trace conversion to the lower (copied) phrase if that phrase is a DP,
4. Remove either the copy or the original term from the linearization.

(Nunes 1996, 1999) had the idea that (10d) might be derivable from the linearization scheme. Suppose, he suggested, that we imagine that the constraints on a linearization algorithm cannot distinguish a thing from its copy. Consider now what would happen in a simple case of phrasal movement like (11).

(11)

(12)

(13)

(14)

(15)
(4) will again interpret all of these orderings as precedence, and this will give us the correct string: a student must run.

In our modification of Kayne’s system we can do without Nunes’ deletion operation. If we let Totality also treat a copy and the thing it is copied from as the same thing, then we are allowed to let A not involve one of those two phrases and the resultant d(A) will simply not contain the ordered pairs that would be generated by that phrase. For instance, if we let A not include the lower DP in (11), then we’ll get (13) directly.

To exploit Nunes’ idea, we should find a way of defining “copy” so that Anti-symmetry and Totality treat a copy as the same thing as the item is copied from. An idea that is being pursued in a number of different ways is that there aren’t really two copies for a moved item, but that there are just two positions that a moved item occupies. That would straightforwardly explain why these constraints on a linearization treat the terminals in a moved phrase as the same thing – the are the same thing.

I think the first person to work this out in detail was Elizabeth Engdahl in her UMass dissertation, but it is being re-explored after a long hiatus by a number of others. One way of expressing the idea uses the following movement theory.

(14) The Remerge Theory of movement
   a. Let \textsc{merge} apply to XPs or X\textsuperscript{0}s in a phrase marker.
   b. Let \textsc{Trace Conversion} give a second meaning to a DP that has more than one sister.

(15) Let \( \sigma \) be one or two things that are either X\textsuperscript{0} or XP.

\[
\textsc{merge}(\sigma) = \text{def.} \text{ replace } \sigma \text{ with } p(\alpha), \alpha \in \sigma, \text{ where }

\begin{align*}
& a. \text{ } p(\alpha) \text{ is an } \alpha^0 \text{ that immediately dominates } \sigma \text{ if } \sigma \text{ is made of two } X^0\text{s, otherwise:} \\
& b. \text{ } p(\alpha) \text{ is an } \alpha P \text{ that immediately dominates } \sigma.
\end{align*}

I’ve made \textsc{merge} do what \textsc{merge} and \textsc{project} previously did. What \textsc{merge} now does is take either one or two things and replaces one of them with a projection of that thing that immediately dominates all of them.

This will give us derivations like (16).

(16) \[
\text{a. } \{ [\iota^0 \text{ must}], [\iota^0 \text{ student}], [\iota^0 \text{ must}], [\iota^0 \text{ AGNT}], [\iota^0 \text{ run}] \}
\]

\[
\text{b. } \{ [\iota^0 \text{ must}], [\iota^0 \text{ student}], [\iota^0 \text{ AGNT}], [\iota^0 \text{ run}] \}
\]

\[
\text{c. } \{ [\iota^0 \text{ must}], [\iota^0 \text{ student}], [\iota^0 \text{ must}], [\iota^0 \text{ AGNT}], [\iota^0 \text{ run}] \}
\]
Let's begin by considering how the semantics of this will work.

Our previous interpretation of how the semantics works with the copy theory of movement was that only one copy of a phrase had to be semantically interpreted, and which that copy is left to the semantics. In this representation, the same idea can be expressed this way:

\[
(\text{one.pdf}) \text{ The denotation of a phrase must be composed with at least one of its sisters.}
\]

The DP in our tree has two sisters, so it need be composed with only one of those. In addition, because this DP has two sisters, it has gained two denotations by way of the trace conversion rule. These two denotations are (18).

\[
(18) \quad \text{a. } \llbracket \text{a student} \rrbracket = \lambda P. \lambda Q. \exists x. P(x) = 1 \land Q(x) = 1 \\
\text{b. } \llbracket \text{a student} \rrbracket = n, \text{if student}(n) = 1
\]

When I say that it has both those denotations, what I mean is that when it composes with its sister, it can do so with either of these two denotations. To show how that idea works here would require me to resolve a difference in our semantics and the one you've learned about quantification. The semantics of quantification we've adopted in concert with our separation of the external \(\theta\)-role from the meaning of the verbal root is that verbs and the phrases they project be predicates of individuals. But the semantics we've adopted in concert with our separation of the external \(\theta\)-role from the meaning of the verbal root is that verbs and the phrases they project are predicates of events. If we wish to continue with the view that VPs are predicates of events, we will have to adapt our denotation for quantifiers. But rather than set off on that mission here, I will instead revert to a view of verbs and their projections that is built for the semantics of quantification that you are familiar with. Assume, then, that VPs and VPs describe individuals. Assume, for instance, that the VP \(\text{run} \) has this denotation:

\[
(19) \quad \llbracket \text{run} \rrbracket = \lambda x. \text{run}(x) = 1
\]

If we decide to use the denotation in (18a) in composing \(\text{a student} \) with VP, then we get (20).

\[
(20) \quad \llbracket \text{run} \rrbracket = \lambda x. \text{run}(x) = 1
\]

Note that we did not compose \(\text{a student} \) with IP. That is because neither of its denotations can compose with the denotation we've given to the IP – there is no semantic composition rule that would put \(\llbracket \text{IP} \rrbracket \) together with either of the meanings in (18). The DP in (20) is therefore interpreted semantically in only its lower position. This gives us the meaning in which \(\text{a student} \) is in the scope of \(\text{must} \).

To get the meaning in which \(\text{a student} \) falls outside the scope of \(\text{must} \) we'll have to use the denotation this DP has by virtue of Trace Conversion. Things don't work entirely smoothly. Let's see what happens if we decide to use (18b), the meaning that comes from Trace Conversion, for the denotation of \(\text{a student} \) when we compose it with VP and do nothing else:
We want to block this meaning. We want the variable that is introduced by Trace Conversion – the $n$ in our formulas – to be obligatorily bound. I don’t know how to do this except by stipulation. This stipulation is going to be equivalent to the stipulation in the Trace theory and Copy theory of movement that requires the moved phrase to bind its trace (or lower copy) when Trace Conversion is invoked. That stipulation can be expressed this way:

(22) If a DP uses its Trace Converted denotation, then its other denotation, \( \llbracket [DP] \rrbracket \), must be combined with another of its sisters, $a$, with this rule:

\[
\llbracket [DP] \rrbracket ( \lambda n. [a] )
\]

where $n$ is the variable introduced by Trace Conversion.

This gives us the meaning we are looking for:

(23)

This gives us the meaning we are looking for:

Now let’s consider how the linearization algorithm will apply. That algorithm, recall, has an engine which generates ordered pairs of terminals and then submits that set of ordered pairs to these constraints:

(24)

a. If $x < y$ and $y < z$, then $x < z$. (transitive).

b. For all distinct terminals $x$ and $y$ in a phrase marker, either $x < y$ or $y < x$. (total).

c. not $(x < y$ and $y < x)$ (antisymmetry).
We’ve added to this the following two (informal) constraints which are intended to convey the headedness parameters.

\[(25) \quad \text{a. Leftwards} \]
\[
\text{If } \alpha \text{ moves and merges to } \beta, \text{ then the left edges of } \beta \text{ and } \alpha \text{ must align.}
\]

\[(25) \quad \text{b. Headedness} \]
\[
\text{If } \alpha \text{ immediately contains } \beta, \text{ and } \alpha \text{ is not a projection of } \beta, \text{ then the left, or right, edges of } \beta \text{ and } \alpha \text{ must align.}
\]

I want to now render these two constraints more formally, and I’ll do so in a way that exploits our new phrase markers. Indeed, what I’m going to do is replace the contribution the LCA made to getting the generalization that “Specifiers come first.” So rather than the LCA I will assume:

\[(26) \quad \text{If } x \text{ and } y \text{ are in a phrase marker, then } x < y \text{ is generable.} \]

We simply allow all orderings of terminals, and we’ll let the constraints in (24) and the versions of (25) I’m about to unveil do the rest.

The formal versions of Leftwards and Headedness are going to express the following idea. In trees of the form (27), the default order is YP precedes everything in its sister.

\[(27) \quad \text{XP} \]
\[
\text{YP XP/}X^{0}
\]

That will make head final languages the default. To do that, I’ll use Kayne’s definition of \(d\).

\[(28) \quad d(\alpha) = \{w: w \text{ a word dominated by } \alpha \} \]

Now the definition:

\[(29) \quad \text{If the mother of } A \text{ is a projection of } A’s \text{ sister, } B, \text{ then } y \notin x, \text{ if } x \in d(A) \text{ and } y \in d(B). \]

This condition prevents any of the words in a phrase adjoined to another from being preceded by the words in its sister. Totality requires that all the words in these two phrases be ordered, however, so this forces all the words in the adjoined phrase to precede all the words in the phrase it has adjoined to. For (30), this will produce the string in (31).

\[(31) \quad \text{student a run AGNT must} \]

That is a possible ordering in the world’s languages, it arises in uniformly head final languages that have movement of subjects.

To get head initial languages, (29) comes with this exception:

\[(32) \quad (29) \text{ unless } x \notin d(B), \text{ in which case } y < x. \]

This will reverse the ordering if the phrase that is adjoined to another isn’t also in the phrase it’s adjoined to. Only moved phrases are in the phrases they are sisters to, so this reversal won’t apply to moved phrases. This will produce from (30) the string in (33).

\[(33) \quad \text{a student must AGNT run.} \]

English!

References

