Movement as higher-order structure building

Patrick D. Elliott (ZAS)
June 11, 2019

Universität Göttingen
Current theories of movement at give rise to conceptual worries vis a vis interface requirements. Is Internal Merge causing more problems than it solves?

The goal here: develop a radically different perspective on syntactic displacement as higher-order structure building, borrowing well-established standard mechanisms from Montagovian semantics for dealing with semantic displacement (i.e., scope).

Some payoffs include:

- No need for trace conversion.
- An account of Müller’s (2001) generalized order preservation.
- An account of the interaction between scrambling and scope-taking in scope-rigid languages such as Japanese.
A (non-standard) overview of movement in minimalist syntax + some conceptual worries.

An analogy between overt syntactic displacement and the QR-analysis of semantic displacement.

Reifying the analogy in a purely derivational system via higher-order structure building.

An analysis of wh-movement.

An extension to quantifier raising and scrambling.

Finish!
Formal Preliminaries
• Since this is a theory talk, let’s try to be precise about the operations we’re using.
• *Types* will help us give an explicit treatment of syntactic operations as *functions*.
• Fortunately, we’re only going to need one primitive type: Let $t$ be the *type* of a Syntactic Object (so). Whenever I talk about syntactic types or variables over sos, I’ll use blackboard font.
• We can’t really do anything interesting with just our primitive type \( t \). We’ll also avail ourselves of function types.
• I’ll use \((\to)\) as the constructor for function types (cf., e.g., Heim & Kratzer 1998 who use \(\langle.\rangle\)).
• \(a \to b\) is the type of a function from things of type \(a\) to things of type \(b\).
• Where Heim & Kratzer write \(\langle\langle e, t\rangle, t\rangle\), i’ll write \((e \to t) \to t\).
• N.b. that \((\to)\) is right-associative, so \(e \to e \to t \equiv e \to (e \to t)\).
• We’ll take as our starting point the hypothesis that the basic structure-building operation in natural language is **Merge** (Chomsky 1995).

• We define **Merge** in a pretty standard way – it’s a *function* that takes two sos, and returns a new (unlabelled) so.

(1) **MERGE** (def.)

\[ \mathcal{X} \ast \mathcal{Y} := [\mathcal{X} \, \mathcal{Y}] \quad \overset{:=}{=} \quad t \rightarrow t \rightarrow t \]

• **Note**: following, e.g., Stabler (1997), we assume that merge is asymmetric:

\[ \mathcal{X} \ast \mathcal{Y} \neq \mathcal{Y} \ast \mathcal{X} \]
• **Merge** successively applies to sos constructing a structured representation, as in (2):

\[
(2) \quad [\text{Andreea [likes Yasu]}]
\]

\[
\star := t \rightarrow t \rightarrow t
\]

\[
\begin{array}{c}
\text{Andreea} := t \\
[\text{likes Yasu}] \\
\star := t \rightarrow t \rightarrow t
\end{array}
\]

\[
\begin{array}{c}
\text{likes} := t \\
\text{Yasu} := t
\end{array}
\]

• **Important**: the tree is a graph of the *derivation*, rather than a representation in its own right.
• Let the type of the atomic unit of syntactic computation (a lexical object, root, etc.), be L. This allows us to define $t$ recursively.

\[
t := L \mid [t]
\]
Movement in MERGE-based frameworks
• Certain expressions (such as *wh*-expressions) are pronounced in positions other than where they’re interpreted – or, more precisely, where a *part* of their meaning (the variable) is interpreted.

• The standard approach to this phenomenon in minimalism is **Internal Merge**.

• This can be cashed out in two different ways: the *copy theory* and the *multidominance theory* of movement.

• I’ll just present the copy theory for exposition, but multidominance approaches are subject to the same issues.
According to the copy theory, movement involves merging a *copy* of an *so* contained within the derived syntactic structure.
It’s not trivial to implement \textsc{Internal Merge} as a function. It should traverse through the constructed syntactic representation for the so to be copied-and-remerged (although see Collins & Stabler 2016 for a local formulation).
Regardless of how **INTERNAL MERGE** is implemented, the representation interpreted by the semantic component must look something like this (Fox 2002, Sauerland 2004):
\( \lambda k \cdot \bigcup_{\text{boy } x} k \ x \ \lambda x' : \text{boy } x' . \{ J \text{ hugs } x' \} \)

which boy

\( \vdash \)

\( i \ldots \)

\( CQ \ldots \)

Josie \ldots

\( \text{hugs } \ i x[\text{boy } x \land x = g_i] \)

THE\(_i\) boy

\begin{align*}
\{ J \text{ hugs } x \mid \text{boy } x \} & \quad (4) \quad \text{\([\text{THE}_i]^g = \lambda P . \ i x[P \ x \land x = g_i]\)} \\
\lambda k \cdot \bigcup_{\text{boy } x} k \ x \ \lambda x' : \text{boy } x' . \{ J \text{ hugs } x' \} & \quad (5) \quad \text{\textsc{Predicate abstraction (def.)}} \\
\text{\([i \times]^g = \lambda x . \ [\times]^{g[i \to x]}\)}
\end{align*}
• How do we get from a copy-theoretic representation to the representation required by the semantics?
• First off, we need a syntactic operation that applies to the lower copy, and replaces the determiner with \( \text{THE}_i \).

\begin{equation}
\text{Trace Conversion (def.)}
\text{tc} \ [\mathcal{D} \mathbb{N}]_i := [\text{THE}_i \mathbb{N}]
\end{equation}

• We also need a syntactic operation that places a binding index immediately below the higher copy, in order to trigger abstraction over the lower copy.
• Due to the demands of the interface, much of the conceptual appeal of *Internal Merge* is lost.

• *Trace Conversion* = the name for a problem, rather than a solution (although, see Fox & Johnson 2016 for a more principled account).

• Goal for the next section: an approach which retains the conceptual appeal of *Internal Merge*, where meaning-computation can proceed in tandem with movement derivations, without the need for syntactic magic, such as *Trace Conversion*, and binding index insertion.
Higher-order structure building
The discussion ahead

- Exploring a (failed?) analogy with between displacement as Quantifier Raising.
- Reifying the analogy in a derivational framework.
- Introducing our players:

**scopal-Merge (★)** Our version of internal merge.

**Lift (↑)** Converting an so into a trivial scope-taker.

**Higher Order Merge (⊗)** A combinatorics for scopal syntactic values.
Before we present our analysis, let’s entertain an analogy.

Imagine that derivation graphs are, themselves, fully-fledged representations.

(7) \[ \text{[Andreea [likes Yasu]]} \]

\[
\begin{array}{c}
\ast \\
\text{Andreea := t} \quad \text{[likes Yasu]} \\
\ast \\
\text{likes := t} \quad \text{Yasu := t}
\end{array}
\]
Now, let’s define a new unary operation, s-MERGE (i.e., *scopal merge*), which we’ll write as (⋆). It’s just defined in terms of merge + lambdas and variables.

\[(8) \quad \star \, \mathbb{X} := \lambda k \cdot \mathbb{X} \ast (k \, \mathbb{X}) \quad (\star) := \mathbb{t} \rightarrow (\mathbb{t} \rightarrow \mathbb{t}) \rightarrow \mathbb{t} \]

• (⋆) takes a so, and shifts it into a function that takes a function from sos to sos, and returns an so.

\[(9) \quad \star \, \text{Andreea} = \lambda k \cdot \text{Andreea} \ast (k \, \text{Andreea}) \quad (\mathbb{t} \rightarrow \mathbb{t}) \rightarrow \mathbb{t} \]

• You can think of (⋆) as a function from an so to something that *takes scope* over sos.
• If we apply (⋆) to an so over the course of our derivation, we end up with a type mismatch. **MERGE** takes two arguments of type \( t \).

\[
\text{Yasu} \times
\]
\[
\star \equiv t \rightarrow t \rightarrow t
\]
\[
t \quad (t \rightarrow t) \rightarrow t
\]
\[
\text{likes} \quad | \quad \star \quad | \quad \text{Andreea}
\]
In order to resolve this type mismatch let’s assume we can scope out the \(\star\)–shifted so via QR – assuming that something of type \((t \rightarrow t) \rightarrow t\) binds a type \(t\) variable.

The result will be a kind of derivational scope; \(\star\) Andreea contributes the so Andreea locally, and the function \((\lambda x . \text{Andreea} \star x)\) takes scope.

When we compute the result, we will end up with a copy-theoretic representation.
An analogy with QR \( v \)

\[
[\lambda k \cdot \text{Andreea } \ast (k \text{ Andreea})] \ (\lambda \times . [\text{Yasu } \text{likes } \times ])
\]

\[= \text{Andreea } \ast ([\lambda \times . [\text{Yasu } \text{likes } \times ]] \ \text{Andreea})\]

\[= \text{Andreea } \ast ([\text{Yasu } \text{likes Andreea}])\]

\[= [\text{Andreea } [\text{Yasu } \text{likes Andreea}]]\]

\[
\lambda k \cdot \text{Andreea } \ast (k \text{ Andreea}) \quad \lambda \times . [\text{Yasu } \text{likes } \times ]
\]

\[
\ast \quad \text{Andreea}
\]

\[
\lambda \times . [\text{Yasu } \text{likes } \times ]
\]

\[
\ast
\]

\[
[\text{Yasu } \text{likes } \times ]
\]

\[
\ast
\]

\[
\text{likes } \times
\]
• Unfortunately, the analogy with QR breaks down – since derivation graphs are not themselves representations, it doesn’t really make sense conceptually to posit an operation of QR that applies to a derivation graph.

• What we want, intuitively, is a way of compositionally integrating scopal values into a computation.

• We’ll model our approach on Barker & Shan’s (2014) continuation semantics.
Tower notation for scopal values

- *Scopal* values are of type \((a \rightarrow b) \rightarrow b\). Barker & Shan (2014) introduce a convenient notational shortcut for scopal types – tower types.

\[ \frac{b}{a} \quad := \quad (a \rightarrow b) \rightarrow b \]

- Similarly, scopal values themselves can be rewritten using tower notation:

\[ \frac{f \, []}{x} \quad := \quad \lambda k . \ f \, (k \, x) \]
• Standard entries for quantificational expressions can be rewritten using tower notation, like so:

\[(12)\] everyone = \(\lambda k \ . \ \forall x[k \ x] \quad \ ::= \ (e \rightarrow t) \rightarrow t\]

\[
\frac{\forall x[\[]}{x} = \frac{t}{e}
\]
We can now rewrite the syntactic operation s-MERGE (⋆) using tower notation:

(13)  \[
\begin{array}{c}
\star \times := \frac{\times \star [\ ]}{\times} \\
\quad \quad (\star) \::= \quad t \\
\end{array}
\]

Recall that ⋆—shifting a so gives rise to a type mismatch in the derivation. Let’s explore a different way of incorporating ⋆—shifted sos into the derivation.
• In order to do this, we need to define two new derivational operations.

• \textsc{Lift} takes an so and returns a \textit{trivially} scopal/higher-order so.

\begin{equation}
(14) \quad \text{\textsc{Lift} (def.)}
\end{equation}

\[
\mathcal{X}^\uparrow := \lambda k . k \mathcal{X}
\]
• **Higher Order Merge** provides us with a way of merging two higher-order/scopal syntactic objects.

\[(15) \text{ Higher Order Merge (def.)} \]

\[m \odot n := \lambda k \cdot m (\lambda X \cdot \lambda n \cdot (\lambda Y \cdot \lambda k \cdot (X \ast Y)))\]
• In order to see what’s going on, it will be easier to rewrite these functions using tower notation.

• LIFT converts an so into a trivial tower.

\[ \Xi^\uparrow := \begin{array}{c} [] \\ \Xi \end{array} \]

• HO MERGE provides a way of merging two towers.

\[ \Xi \odot \begin{array}{c} f [] \\ \Xi \end{array} \quad \begin{array}{c} g [] \\ \Xi \end{array} := \begin{array}{c} f [g []] \\ \Xi \end{array} \quad \begin{array}{c} \Xi \ast \Xi \\ \Xi \ast \Xi \end{array} \]
• Now we have everything we need to incorporate $\star$—shifted sos into the syntactic derivation:

```
Yasu $\star$ []
|  
[Andreea [likes Yasu]]
|  
⊗

/   \\
[]     []
/   \\
Andreea          Yasu $\star$ []
|    |        
|    |        
Andreea↑         ⊗

/   \\
[]     []
/   \\
likes     Yasu
|    |        
|    |        
likes↑    $\star$ Yasu
```
• Finally, we need a syntactic operation to *lower* a higher-order so back down to an ordinary so. We can define `Lower` simply as the identity function.

\[
\text{(16) } \text{Lower (def.)} \\
\downarrow m := m \text{id}
\]

• *Lowering* the higher-order so gives us the same result as the copy theory of movement!

\[
\downarrow \left( \frac{\text{Yasu} \ast [\ ]}{[\text{Andreea} [\text{likes Andreea}]]} \right) = [\text{Yasu} [\text{Andreea} [\text{likes Andreea}]]]
\]
• Since we’re adopting a radically derivational perspective, we don’t really need to refer to the outputted structural representations for anything. Let’s simplify things and just treat \textsc{Merge} as \textit{concatenation} (see Kobele 2006 for a thorough demonstration that this is harmless).

\[(17) \quad \text{𝕏} \ast \text{Ỹ} := \text{𝕏} : \text{Ỹ}\]

• On this view, it’s natural to redefine $\star$ such that the local value has null phonological content:

\[(18) \quad \star \text{𝕏} := \frac{\text{𝕏} \ast []}{\emptyset}\]
(19) Yasu, Andreea likes.

(20) \(((\text{Andreea}^\uparrow) \odot (\text{likes}^\uparrow) \odot (\star \text{Yasu})))^\dagger

= (\lambda k . \text{Yasu} \ast (k (\text{Andreea} : \text{likes} : \emptyset)))^\dagger

= \text{Yasu} : \text{Andreea} : \text{likes} : \emptyset
Extension to wh-movement
Incorporating a basic feature calculus

- In order to extend the proposal to *wh*-movement, we must make it more syntactically realistic. We’ll treat sos as *feature bundles*; *MERGE* concatenates feature bundles.
- We can now redefine *merge* as a *feature sensitive* operation.
- Merging an so \( \times \) with an uninterpretable \( Q \) feature with another so \((Υ : ℤ)\) results in *ungrammaticality* (♯), unless the head \( Υ \) carries an interpretable \( Q \) feature.

\[(21) \quad \begin{align*}
\text{a.} & \quad \times_{uQ} * (Υ_{iQ} : ℤ) = \times : Υ_{iQ} : ℤ \\
\text{b.} & \quad \times_{uQ} * Υ = ♯ \\
\text{c.} & \quad \times * Υ = \times : Υ
\end{align*}\]

- This needs to be generalised, but this will do for now.
• We can now additionally redefine (⋆) in a feature sensitive way:

\[(22) \quad \star \mathcal{X}_{[d,uQ]} := \mathcal{X}_{[d,uQ]} / \emptyset_{[d]} \]

• We now have everything we need to account for feature-driven movement in a more realistic way:

\[(23) \quad \text{a. } ((C_{[iQ]}^\uparrow) \otimes ((Andreea^\uparrow) \otimes ((\text{likes}^\uparrow) \otimes (\star \text{who}_{[d,uQ]}))))^\downarrow \]

\[\text{b. } (\lambda k . \text{who}_{[d,uQ]} * k (C_{[iQ]} : \text{Andreea} : \text{likes} : \emptyset_{[d]}))^\downarrow \]

\[\text{c. } \text{who}_{[d]} : C_{[iQ]} : \text{Andreea} : \text{likes} : \emptyset_{[d]} \]
• order preservation effects are pervasive in syntax (Müller 2001), e.g., superiority effects in English and multiple wh-fronting languages such as Bulgarian.

(24) a. I wonder who^x t_x bought what^y.
   b. *I wonder what^y who bought t_y.

(25) a. Koj kakvo kupuva?
       Who what buys?
   b. *Kakvo koj kupuva?
       What who buys?
• Order-preservation falls out as the unmarked case in the system outlined here. This is because HO MERGE (repeated below) sequences movements from left-to-right.

\[
\begin{align*}
&f [] \circ g [] := f [g []] \\
&\text{x} \quad \text{Y} := \text{x} \ast \text{Y}
\end{align*}
\]

• We ignore the feature calculus here for ease of exposition:

(26) a. \( \downarrow (((\ast \text{who}) \circ ((\text{buys}^\dagger) \circ (\ast \text{what})))) \)

b. \( =\downarrow (\lambda k . \text{who} \ast (\text{what} \ast (k (\emptyset : \text{buys} : \emptyset)))) \)

c. \( = \text{who} : \text{what} : \emptyset : \text{buys} : \emptyset \)
• In this system, semantic computation can proceed in tandem with syntactic computation. We’ll assign a single meaning to a *wh*-expression which will predict that it scopes exactly at the position it’s moved to.

• We adopt a generalized Karttunen semantics for *wh*-expressions – they scope over question meanings and return question meanings (Cresti 1995, Charlow 2014, Elliott 2017)

\[
\begin{align*}
\llbracket \text{who} \rrbracket := \lambda k . \bigcup_{\text{person } x} k x & \quad (e \rightarrow \{ t \}) \rightarrow \{ t \} \\
\end{align*}
\]

• Note that *wh*-expressions have a *scopal* semantics – we can scope them using semantic correlates of *Lift* and HO *Merge* (Barker & Shan 2014).
Return and Scopal Function Application

- We take the semantic correlate of the syntactic operation \textsc{Lift} to be \textsc{Return} ($\rho$).

\[
(28) \quad x^\rho := \frac{[]} x \quad \quad (\rho) ::= a \to (a \to \{b\}) \to \{b\}
\]

- We take the semantic correlate of the syntactic operation \textsc{HO Merge} to be \textsc{Scopal Function Application} ($S$).

\[
(29) \quad \frac{f [] \quad g []}{S} \quad \frac{x \quad y}{A \ x \ y} := f [g []]
\]
Finally, we take the meaning of $C_{[iQ]}$ to be singleton-set formation.

$$\{ \text{Andreea likes } x \mid \text{person } x \}$$
• Note the isomorphism between the semantic computation and syntactic computation. Both are computed step-by-step, in tandem.

(30)  a. Syntax:
\[
\left[ C_{iQ} \right] \left( \left( \left[ \text{Andreea} \right] \right) \left( \left[ \text{likes} \right] \right) \left( \left[ \text{who}_{d,uQ} \right] \right) \right)
\]

b. Semantics:
\[
\left( \left( C_{iQ} \right) \left( \text{Andreea} \right) \left( \text{likes} \right) \left( \text{who}_{d,uQ} \right) \right)
\]

• There is no need for anything like trace conversion. In the syntax, movement corresponds to scoping the features + phonological content of a syntactic object, in the semantic component, anything with a scopal semantics exhibits *interpretive* displacement via the same mechanisms.
• **Merge** in the syntax corresponds to **Function Application** in the semantics: \((*) \approx A\)

• When a moved expression is **scopal** (i.e. interpreted in its derived position):
  • **Lift** in the syntax corresponds to **Return** in the semantics (in fact, they’re polymorphic instantiations of the same function): \((\uparrow) \approx (\rho)\)
  • **HO Merge** in the syntax corresponds to **Scopal Function Application** in the semantics: \((\otimes) \approx S\)
Quantifier Raising

• In this system, quantifier raising simply involves a scopal semantics with a non-movement syntax. There is in fact no need for covert movement.

(31) a. Syntax:

some linguist ∗ (hates ∗ [every philosopher])
= [some linguist] : hates : [every philosopher]

b. Semantics:

\[
\begin{align*}
&\left( \exists y [\text{linguist } y \land \left[ \right] ] \right) S \left( \left[ \right] S \forall x [\text{phil } x \rightarrow \left[ \right] ] \right) \downarrow \\
&\quad = \exists y [\text{linguist } y \land \forall x [\text{phil } x \rightarrow y \text{ hates } x]]
\end{align*}
\]

• Note that the unmarked case in this system is surface scope. This is a good prediction for scope-rigid languages like German, but we need to do a little more to get inverse scope.
Barker & Shan (2014) show that we can derive inverse scope by *internally lifting* (↑↑) the lower quantifier, and (re-)lifting the higher quantifier. Details suppressed here but see Barker & Shan.

\[
\begin{align*}
\left( \begin{array}{c}
\exists y [\text{ling } y \land []] \\
\end{array} \right) & \quad S \quad \left( \begin{array}{c}
\exists y [\text{ling } y \land []] \\
\end{array} \right) \\
\left( \begin{array}{c}
\forall x [\text{phil } x \rightarrow []] \\
\end{array} \right) & \quad S \quad \left( \begin{array}{c}
\forall x [\text{phil } x \rightarrow []] \\
\end{array} \right) \\
\left( \begin{array}{c}
\exists y [\text{ling } y \land y \text{ hates } x] \\
\end{array} \right) & = \quad \forall x [\text{phil } x \rightarrow \exists y [\text{ling } y \land y \text{ hates } x]]
\end{align*}
\]
Let’s assume that internal lift is freely available in English, without any syntactic reflex. This predicts the availability of scopal ambiguities.

Languages such as Japanese and Hindi are ordinarily scope rigid however; scopal ambiguities may arise if a scopal expression is scrambled.

(32) a. Dareka-ga daremo-o sonkeisiteiru
    someone-NOM everyone-ACC admire
    some > every, *every > some

b. daremo-o dareka-ga t sonkeisiteiru
    everyone-ACC someone-NOM t admire
    some > every, every > some
There’s a very natural perspective to adopt in languages such as Japanese and Hindi – *internal lift* isn’t freely available, rather, it is the semantic reflex of s-MERGE (★).
(33) **Syntax:**
\[
([\text{Some philosopher}]^\uparrow \otimes ([\text{hates}]^\uparrow) (\star [\text{every linguist}])))^\downarrow
\]
\[
= [\text{every linguist}] : [\text{some philosopher}] : [\text{hates}] : \emptyset
\]

(34) **Semantics:**
\[
(([[\text{some philosopher}]^\rho] S ([[\text{hates}]^\rho \circ^\rho] S ([\text{every linguist}]^\uparrow))))^\downarrow
\]
\[
= \begin{pmatrix}
[\text{every linguist}] \lambda x [ ]
\end{pmatrix}^\downarrow
\]
\[
= \begin{pmatrix}
[\text{some philosopher}] \lambda y [ ]
\end{pmatrix}
\]
\[
y \text{ hates } x
\]
\[
= \forall x[\text{ling } x \rightarrow \exists y[\text{phil } y \land y \text{ hates } x]]
\]
• But scrambling doesn’t just give rise to inverse scope – it gives rise to *scopal ambiguities*.

• We can account for this by simply positing an *implicational* rather than a *one-to-one* relationship between (⇧⇧) and (⋆) – (⇧⇧) (in Japanese) implies (⋆) in the syntactic computation, but not vice versa.

• In other words, (⇧⇧) is an *optional* semantic reflex of (⋆), but it is not permitted in the absence of (⇧⇧).
Conclusion
future prospects

• How to account for the following within this framework:
  • *locality* – has a natural treatment in terms of obligatory lowering; see Charlow (2014) on scope islands.
  • *Successive-cyclicity* – has a natural treatment in terms of *lowering* followed by re-s-MERGEing.
  • *Reconstruction* – see Barker & Shan (2014) for a detailed treatment consistent with this system.
  • *Late merge* – more difficult, but can be analyzed without copies once more sophisticated mechanisms for scope-taking (*indexed continuations*) are adopted. I’ll come back to this in future work.
• In this talk, I’ve suggested that we can take a cue from the formal semantics literature, and treat syntactic displacement as a kind of *syntactic* scope-taking.

• This move has a major conceptual advantage – semantic computation can proceed *in tandem with* syntactic computation. There is no need for any *ad-hoc* mechanism for interpreting movement.

• We’ve mentioned a couple of interesting empirical payoffs – the analysis of generalised order preservation, and scrambling.

• A more thorough exploration of the properties of this system will have to wait for another time!
Thanks for listening!


