A FORMAL FOUNDATION FOR A AND A-BAR MOVEMENT

It is natural to think of sentences in natural language as exhibiting long-distance dependencies. A surprising fact, established over decades of linguistic investigation, is that such dependencies are naturally treated as falling into one of two groups. 'A' dependencies (typified by raising, passivization, etc) are those which allow for re-binding, which disallow reconstruction, and which do not license parasitic gaps. 'A-bar' dependencies (typified by wh-movement, relativization, etc) disallow re-binding, allow reconstruction, and do license parasitic gaps. Furthermore, when an expression enters in to both A and A-bar dependencies, all of its A dependencies must 'precede' its A-bar dependencies [2]. This relational property of dependencies is known as the ban on improper movement. In the government and binding (GB) tradition, all of these properties of these two dependency types must be independently stipulated—none follow from any of the others. Here I will present a simple and constrained formal system in the GB tradition with two kinds of long distance dependency forming operations, the interaction of which gives rise to the major syntactic property of A and A-bar movement: the ban on improper movement. Additionally, the resulting system seems to provide the right kinds of structures over which to naturally enforce the other characteristic semantic properties (reconstruction and re-binding) of each kind of movement. The difference between A and A-bar dependencies as regards the licensing of parasitic gaps cannot be made to follow from the formal architecture of the system, as neither do parasitic gaps. However, natural grammars can be written for fragments of English in which the difference in licensing of parasitic gaps obtains without stipulation.

1. Minimalist grammars

Minimalist grammars [6] are a midly context-sensitive grammar formalism embodying some of the core ideas of Chomsky's minimalist program. Expressions are finite sequences of pairs consisting of exponents (strings) and finite sequences of features, and there is a single binary operation ('merge') and a single unary operation ('move'). Both operations are conditioned solely on the basis of the features in their arguments, and both simplify their arguments in the sense that the number of features contained in the result is strictly less than the sum of the number of features contained in the arguments. Formally, a minimalist grammar over an alphabet Σ is given by a tuple $G = \langle \mathbf{lic}, \mathbf{sel}, Lex \rangle$, where **lic** and **sel** are finite sets of licensing and selection feature types, and $Lex \subseteq_{fin} \Sigma \times \mathbb{F}^*$ is a finite set of atomic

expressions, where $\mathbb{F} := \{\texttt{+f}, \texttt{-f}, \texttt{=s}, \texttt{s} : f \in \mathbf{lic}, s \in \mathbf{sel}\}$. An expression is a finite sequence $\phi_0, \ldots, \phi_n$, where $\phi_i \in \Sigma^* \times \mathbb{F}^*$, $0 \leq i \leq n$.

The definition of the merge operation is broken into two cases depending upon the shape of its second argument:

$$\frac{(\sigma, \texttt{=s}\delta), \phi_1, \ldots, \phi_n \qquad (\tau, \texttt{s}), \psi_1, \ldots, \psi_k}{(\sigma\tau, \delta), \phi_1, \ldots, \phi_n, \psi_1, \ldots, \psi_k} mergeA$$

$$\frac{(\sigma, \texttt{=s}\delta), \phi_1, \ldots, \phi_n \qquad (\tau, \texttt{s}\gamma), \psi_1, \ldots, \psi_k}{(\sigma, \delta), \phi_1, \ldots, \phi_n, (\tau, \gamma), \psi_1, \ldots, \psi_k} mergeB$$

Move is similarly partitioned into cases:

$$\frac{(\sigma, \texttt{+f}\delta), \phi_1, \ldots, \phi_{i-1}, (\tau, \texttt{-f}), \phi_{i+1}, \ldots, \phi_n}{(\tau\sigma, \delta), \phi_1, \ldots, \phi_{i-1}, \phi_{i+1}, \ldots, \phi_n} moveA$$

$$\frac{(\sigma, \texttt{+f}\delta), \phi_1, \ldots, \phi_{i-1}, (\tau, \texttt{-f}\gamma), \phi_{i+1}, \ldots, \phi_n}{(\sigma, \delta), \phi_1, \ldots, \phi_{i-1}, (\tau, \gamma), \phi_{i+1}, \ldots, \phi_n} moveB$$

A derivation of an expression e is an at most binary branching tree whose nodes are labelled with expressions, and where the following conditions obtain:

(1) leaves are labelled with lexical items
(2) for every unary branching node f with unique child e, $\langle e, f\rangle \in move$
(3) for every binary branching node f with left child g and right child e, $\langle g, e, f\rangle \in merge$

A well-formed derivation is one in which every node is labelled with an expression $\phi_0, \ldots, \phi_n$ satisfying

(SMC) $$\delta_j \approx_0 \delta_i \text{ iff } j = i$$

where $\phi_k = (\sigma_k, \delta_k)$, and $a\alpha \approx_0 b\beta$ iff $a = b$. Note that, as $|\mathbf{lic}|$ is finite, the SMC bounds the length of any expression occurring in a well-formed derivation at $|\mathbf{lic}| + 1$.

2. Hypothetical reasoning

Features in minimalist expressions can be thought of as resources [5], and there is an intuitive relation between selection features (`=s`) and implicational types in resource logics ($A \multimap B$). (This intuition has been made explicit in the translations of Vermaat [7] and Amblard [1].) The typical long distance dependency dealt with in linguistic theory involves an overt expression dependent upon another, structurally lower, one.[1] Although we have previously been mediating such dependencies by introducing the dependent expression in its lowest such position, and then checking off its features as other positions it is dependent upon become available, another, natural, approach would involve introducing the dependent expression only in its highest, surface, position, and checking its entered-into dependencies

[1]The desired notion of 'lowness', c-command in the derivation tree, is more difficult to state over a derived tree representation. See Kracht [4].

en masse. From our bottom-up perspective, we achieve this by an operation which temporarily eliminates features from an expression, thereby 'freeing up' other features, which were previously hidden due to the list-like nature of the feature data structure. Temporarily eliminated features are 'stored' in an $|\mathbf{lic}|$-ary array $\vec{a}$ (a function from **lic** to $\mathbb{F}^*$). Given an array $\vec{a}$, $\vec{a}_i$ denotes the object in the i^{th} cell and $\vec{a} +_i c$ is the array $\vec{b}$ like $\vec{a}$ except that $\vec{b}_i = \vec{a}_i^\frown c$. For $\vec{a}$ an array, and $i, j \in \mathbf{lic}$, $\vec{a}^{\langle i,j \rangle}$ is identical to $\vec{a}$, but that $\vec{a}_j^{\langle i,j \rangle} = \vec{a}_i$ and $\vec{a}_i^{\langle i,j \rangle} = \vec{a}_j$. Given two arrays, $\vec{a}$ and $\vec{b}$ such that $\vec{a}_i \neq \epsilon \Rightarrow \vec{b}_i = \epsilon$, we define $\vec{a} \oplus \vec{b}$ such that $(\vec{a} \oplus \vec{b})_i = \vec{a}_i^\frown \vec{b}_i$.

$$\frac{(\sigma, \texttt{=s}\delta); \vec{a}}{(\sigma, \delta); \vec{a} +_j \texttt{s}} hypA$$

for some j such that $\vec{a}_j = \epsilon$.

$$\frac{(\sigma, \texttt{+i}\delta); \vec{a}}{(\sigma, \delta); (\vec{a} +_i \texttt{-i})^{\langle i,j \rangle}} hypB$$

for some j such that $\vec{a}_j = \epsilon$.

Elements of the array $\vec{a}$ (hypotheses), are discharged once an expression is merged which has the appropriate feature sequence.

$$\frac{(\sigma, \texttt{=s}\delta); \vec{a} \qquad (\tau, \texttt{s}); \vec{b}}{(\sigma\tau, \delta); \vec{a} \oplus \vec{b}} dischargeA$$

$$\frac{(\sigma, \texttt{+i}\delta); \vec{a} \qquad (\tau, \gamma\texttt{-i}); \vec{b}}{(\tau\sigma, \delta); \vec{c} \oplus \vec{b}} dischargeB$$

where $\vec{a}_i = \gamma$ and $\vec{c}$ is like $\vec{a}$ except that $\vec{c}_i = \vec{b}_i = \epsilon$.

Derivation trees are defined analogously to the merge and move system. Given a lexicon Lex the string language associated with Lex by a minimalist grammar with hypothetical reasoning is defined to be the set of string components of the roots of well formed derivation trees. It is straightforward to verify that the current system is strictly less expressive than the merge and move system: given a lexicon Lex, the string language associated with that lexicon by a minimalist grammar with hypothetical reasoning is context-free.

3. Combining movement and hypothetical reasoning

As defined in the previous section, merge and discharge are two sides of the same coin: merge inserts an expression, with features still to be checked ($f\beta$), and discharge inserts an expression, whose features have already been checked (αf). A natural generalization of both of these operations is to insert expressions, some of whose features have already been checked, and the remaining features of which must still be checked ($\alpha f \beta$).

The definition of the operation *insert* is given in four cases. The first two ($insertA$ and $insertB$) can be viewed either as the merge operation extended

over expressions with feature arrays, or, alternatively, as the first case of the discharge operation extended over expressions with moving pieces.

$$\frac{(\sigma, \texttt{=s}\delta), \phi_1, \ldots, \phi_n; \vec{a} \qquad (\tau, \texttt{s}), \psi_1, \ldots, \psi_k; \vec{b}}{(\sigma\tau, \delta), \phi_1, \ldots, \phi_n, \psi_1, \ldots, \psi_k; \vec{a} \oplus \vec{b}} insertA$$

$$\frac{(\sigma, \texttt{=s}\delta), \phi_1, \ldots, \phi_n; \vec{a} \qquad (\tau, \texttt{s}\beta), \psi_1, \ldots, \psi_k; \vec{b}}{(\sigma, \delta), \phi_1, \ldots, \phi_n, (\tau, \beta), \psi_1, \ldots, \psi_k; \vec{a} \oplus \vec{b}} insertB$$

The final two cases are similar to the second case of the discharge operation. The final case ($insertD$) is the most interesting, making use of both hypothetical reasoning and actual movement.

$$\frac{(\sigma, \texttt{+i}\delta), \phi_1, \ldots, \phi_n; \vec{a} \qquad (\tau, \alpha\texttt{-i}), \psi_1, \ldots, \psi_k; \vec{b}}{(\tau\sigma, \delta), \phi_1, \ldots, \phi_n, \psi_1, \ldots, \psi_k; \vec{c} \oplus \vec{b}} insertC$$

where $\vec{a}_i = \alpha$ and $\vec{c}$ is like $\vec{a}$ except that $\vec{c}_i = \vec{b}_i = \epsilon$, and for no $\phi_j = (\sigma_j, \delta_j)$ or $\psi_j = (\tau_j, \gamma_j)$ does δ_j or γ_j begin with `-i`.

$$\frac{(\sigma, \texttt{+i}\delta), \phi_1, \ldots, \phi_n; \vec{a} \qquad (\tau, \alpha\texttt{-i}\beta), \psi_1, \ldots, \psi_k; \vec{b}}{(\sigma, \delta), \phi_1, \ldots, \phi_n, (\tau, \beta), \psi_1, \ldots, \psi_k; \vec{c} \oplus \vec{b}} insertD$$

where $\vec{a}_i = \alpha$ and $\vec{c}$ is like $\vec{a}$ except that $\vec{c}_i = \vec{b}_i = \epsilon$, and for no $\phi_j = (\sigma_j, \delta_j)$ or $\psi_j = (\tau_j, \gamma_j)$ does δ_j or γ_j begin with `-i`.

Movement and hypothetical reasoning are as before (but extended with feature arrays and moving expressions respectively).

$$\frac{(\sigma, \texttt{+j}\delta), \phi_1, \ldots, \phi_{i-1}, (\tau, \texttt{-j}), \phi_{i+1}, \ldots, \phi_n; \vec{a}}{(\tau\sigma, \delta), \phi_1, \ldots, \phi_{i-1}, \phi_{i+1}, \ldots, \phi_n; \vec{a}} moveA$$

where $\vec{a}_j = \epsilon$.

$$\frac{(\sigma, \texttt{+j}\delta), \phi_1, \ldots, \phi_{i-1}, (\tau, \texttt{-j}\gamma), \phi_{i+1}, \ldots, \phi_n; \vec{a}}{(\sigma, \delta), \phi_1, \ldots, \phi_{i-1}, (\tau, \gamma), \phi_{i+1}, \ldots, \phi_n; \vec{a}} moveB$$

where $\vec{a}_j = \epsilon$.

$$\frac{(\sigma, \texttt{=s}\delta), \phi_1, \ldots, \phi_n; \vec{a}}{(\sigma, \delta), \phi_1, \ldots, \phi_n; \vec{a} +_i \texttt{s}} hypA$$

for some i such that $\vec{a}_i = \epsilon$.

$$\frac{(\sigma, \texttt{+i}\delta), \phi_1, \ldots, \phi_n; \vec{a}}{(\sigma, \delta), \phi_1, \ldots, \phi_n; (\vec{a} +_i \texttt{-i})^{\langle i,j \rangle}} hypA$$

for some j such that $\vec{a}_j = \epsilon$, and for no $\phi_k = (\sigma_k, \delta_k)$ does δ_k begin with `-i`.

Derivation trees are defined as before. Although adding hypothetical reasoning to the standard minimalist grammar framework increases the strong generative capacity of the formalism, the weak generative capacity is unchanged (i.e. $\mathcal{L}(MG) = \mathcal{L}(MG^{hyp})$).

Given this setup, we identify hypothetical reasoning with A movement, and movement with A-bar movement. The ban on improper movement is an

immediate and simple consequence of the fact that hypothetical reasoning takes place before an expression is inserted, and movement after.

4. Conclusions

The fact that there is a sharp dividing line between the multiple dependencies an expression may enter into provides the strong generative capacity necessary to obtain a non-stipulative account of the semantic asymmetries between A and A-bar movement. Differences between A and A-bar movement such as the ability of A moved, but not A-bar moved items to bind moved-over expressions can be recovered via a semantic interpretation scheme that simply binds all and only expressions within the scope of an expression when it is first inserted into the structure. And differences with respect to 'reconstruction' (A-bar movement allows this, A movement does not) is implementable in multiple, interestingly different, ways. First is to follow the suggestion of Chomsky [3], according to which reconstruction is mediated by interpreting an expression in one of its moved-through positions. As our expressions have only ever been present in A-bar positions, we can only interpret them there. Another option, which makes more use of the derivational ambiguity now inherent in the syntactic calculus, is to require that expressions are interpreted in the position in which they are inserted. Reconstruction then indicates that an expression was inserted into a (particular) non-surface position, and thus also ties together the binding mechanism with the scope taking mechanism.

While it is logically possible to concoct a semantics which yields the exact opposite pattern, it is most natural, given the architecture of our syntactic formalism, to group together reconstructability and the inability to rebind variables with A-bar movement, and non-reconstructability and the ability to rebind variables with A movement. Thus we have a system which appears able to make sense of the fact that the two types of long-distance dependencies in natural languages have the clusters of properties they do.

References

[1] M. Amblard. *Grammaire minimaliste catégorielle et interface syntaxe/sémantique*. PhD thesis, Université Bordeaux I, in progress.
[2] N. Chomsky. *Lectures on Government and Binding*. Foris, Dordrecht, 1981.
[3] N. Chomsky. *The Minimalist Program*. MIT Press, Cambridge, Massachusetts, 1995.
[4] M. Kracht. Syntax in chains. *Linguistics and Philosophy*, 24(4):467–529, August 2001.
[5] C. Retoré and E. P. Stabler. Resource logics and minimalist grammars. *Research on Language and Computation*, 2(1):3–25, 2004.
[6] E. P. Stabler. Derivational minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer-Verlag, Berlin, 1997.
[7] W. Vermaat. The minimalist move operation in a deductive perspective. *Research on Language and Computation*, 2(1):69–85, 2004.