’Movement’, precedence, c-command

Michael Brody, UCL, HAS

1d syntax

In one-dimensional syntax we generate strings that in the standard frameworks would correspond to paths from the initial symbol of the tree to (abstract or lexical) terminal elements. This is the p-string, linearly ordered, I assume, by precedence.

Taking the so called cartographic research to what seems like one of its possible logical conclusions, I assume that whatever constraints and principles entail the order of elements in functional sequences (henceforth c-strings) are not additional constraints on trees created by an independent operation of merge but define syntactic objects. C-strings are concatenations of syntactic primitives, themselves concatenatable further.

Taking the initial symbol S to be a p-string, let us define a p-string recursively as in (1):

(1) The concatenation of an initial segment of a p-string with a c-string is a p-string ----where (a) a segment of a string is a single continuous substring and (b) a substring is initial iff it contains the initial symbol

C-strings always end with an (abstract or lexical) terminal element. (1) ensures that this property is inherited by p-strings.

Consider „internal merge”, at least for presentational purposes here, multiple attachment. This will be possible if we allow the initial segment of a p-string P to be concatenated with a c-string that is not provided externally but is a segment of P. Since in most of the cases the top node of the reconcatenated segment will not correspond to the highest category in its c-string, we need to allow the concatenation of final segments of c-strings to the initial segments of a p-string to licence p-strings. Accordingly, revise (1) as (1’):

(1’) The concatenation of an initial segment of a p-string with the final segment of a c-string is a p-string, --where a segment of a string is a single continuous substring that is initial iff it contains the initial symbol and final iff it contains a terminal element.

I leave it open here if (p-string-) externally provided c-strings are always maximal or whether these may also be properly final segments of maximal c-strings.

For a concrete example let’s take „The clown cried”. The concatenatation of the c-string of *cry* with the initial symbol is a p-string, P1, and so is the concatenation of the c-string, of *clown*, CS, with the initial segment of P1, the result is P2. Suppose, this concatenation is „VP-internal”. P2, will then licence another shorter p-string, P3. P3 is the concatenation of the same element, CS, with a shorter initial segment of P2, (which is also an initial segment of P1). This shorter initial segment of P2 ends with the node whose surface daughter the subject is standardly taken to be. Finally the c-string of the article *the* concatenates to a node which is internal to CS. There are two p-strings, P2 and P3, that end with CS. Since CS is the same element in P2 and P3, concatenating with one entails concatenating also with the other. Hence we end up with two p-strings for the article, and generally for any element syntactically preceded by (in standard terms: included in) the initial element of a moved c-string.

Nodes in p-strings can be targets of lexical insertion. While lexical items may span a number of nodes, at most one lexical item can attach to any single node. If we generalize this, so that at most a single element, lexical item or c-string, can be concatenated to a node of a p-string, then we have ensured the effects of binary branching. In fact if we assume that c-strings are constructed in a presyntactic lexicon

(Starke 2010), and only their concatenation ie. (1') is narrow syntax proper, then binary branching is a consequence of the existence of these two components, in both of which only a single element can concatenate to a given element.

The basic assumption of one-dimensional syntax is that syntax sees only the linear orders of individual p-strings. The set of p-strings, the structure of the sentence, is accessible only for the interpretive systems. I assume that the input that syntax provides to the interpretive moduls is a set of precedence statements, the conjunction of the precedence statements of the individual p-strings.

A constituent for semantic interpretation can be recursively defined as a category together with the constituents it precedes.

Spellout, the initial step in phonological interpretation organizes syntactic nodes into a single linear string. It respects syntactic precedence orders and also the constituency structure, but without making reference to the latter. As discussed in Brody 2017, if B and C are sisters then B may precede or follow C, but in general B cannot interrupt C--that is B cannot intervene between constituents of C. Since spellout respects syntactic precedence relations, if A precedes B in syntax then A precedes B in spellout. Given that syntax provides a partial ordering for the interpretive moduls, we cannot assume in general that if A immediately precedes B in syntax then A must immediately precede B in spellout. The set of syntactic structures will generally require A to immediately precede two elements, but A cannot immediately precede more than one element in the linear order of the spellout string. But suppose we restrict the scope of 'immediacy' to categories not preceded by A, that is we disallow only interveners that A does not precede. We can then assume that spellout conservation of syntactic precedence relations is weaker than conserving syntactic immediate precedence, but stronger than conserving only precedence:

(2) If A immediately precedes B in syntax then X can intervene between A and B in spellout only if A precedes X in syntax.

(2) ensures without direct reference to the interpretive notion of constituent that sister nodes cannot interrupt each other, and therefore that their spellout order must respect constituency relations.

Multiple attachment structures cannot be expressed by a single spellout string without contradiction, an element cannot be in two positions in a linear order. Hence either one of the attachments will be missing or the interpretation needs to create two nodes instead of the one that is multiply attached in syntax. One of these may be a copy of some features of the node but more generally it will be silent. Similar resolution of multiple attachment may be necessary in semantic interpretation for different reasons. Here the exigencies of semantics will determine the content of the two constituents that need to be created from the syntactically multiply attached one.

2. Reconcatenate and c-command

Syntactic movement targets positions that both precede and c-command the launching site. We might expect one-dimensional syntax to help us understand why this is the case. The generalisation that links the precedence and the c-command property of movement structures is rather elusive in non-antisymmetric frameworks (Abels and Neeleman 2012, Brody 2015).

An explanation in terms of a weaker version the present framework (precedence syntax) was tentatively suggested earlier (Brody 2015, 2017), based on the requirement that the address of the filler (the node immediately preceding it), must precede (the address of) the gap. Assuming that the condition holds both in syntax and at spellout, successfully captured the generalization.

This approach does not seem ambitious enough. While it captures the precedence -- c-command generalization, it needs to rely on the additional precedence requirement to ensure the existence of these movement properties. As was emphasized in earlier work, it exploits only the assumption that domination relations are in fact precedence relations, but does not rely on the core idea of one-dimensionality: that syntactic structures are strings.

However it is precisely the string –like nature of syntax that generates the expectation that these movement properties might turn out to be non-stipulative, necessary features of the system.

Deriving the c-command requirement is straightforward. In the present framework, syntax has access only to elements of the linear orders internal to p-strings. Hence it will be impossible to concatenate the final c-string of a given p-string to the initial sequence of another, distinct p-string. When we concatenate to the initial segment of a p-string a segment that is not a final segment of a c-string defined by the syntax-external moduls, then we can only reconcatenate the final segment of a given p-string to an initial segment of the same p-string. It follows that the node to which the moved element is reconcatenated, the reconcatenation site, RCS, will syntactically precede the original concatenation site, OCS. In other words movement can only land in a c-preceding position, corresponding to standard c-command.

3. Reconcatenate and spellout precedence

The fact that the RCS will necessarily precede the OCS, does not entail however that the re-concatenated c-string will also precede OCS in the spellout structure. Our assumption about conservation of syntactic precedence in the spellout string stated in (2) says nothing about the order of sisters. Given reconcatenation, RCS will be a binary branching node, and (2) allows the nodes, that RCS immediately precedes to be in either order with respect to each other.

We could short-circuit the problem in the following way. Suppose the concatenation of a final c-string segment starting with C, to an initial segment of a p-string p, the initial segment of p ending with P, (as in (1/1') above) always involves an immediate precedence relation between P and C, at the juncture of the two strings involved. Spellout will inherit this strong immediate precedence requirement from syntax, just like it inherits the weaker immediate precedence requirement of (2) in all other syntactic precedence cases. As a consequence, C will always precede its sister S, the node that follows P in p. Thus, the fact that a reconcatenated c-string segment precedes its OCS would be due to such a c-string being a specifier that, like all specifiers must precede its sister. (The sister S of this c-string in turn must precede the OCS of the c-string, since OCS is an element that follows S in the original p-string p.)

This would in effect be a version of Kayne's (1994) antisymmetry hypothesis: c-strings can be interrupted by their specifiers (including phrasal specifiers and standard XP-heads under this concept) but specifiers must invariably precede their sisters/complements. We could rationalize the strong immediate precedence requirement on concatenation junctures by noting that c-strings as opposed to p/c-string junctures are presyntactically given and therefore recoverable even when their member nodes do not exibit strict adjacency.

As argued earlier the fact that specifiers, whether phrasal or not, of nominal and verbal c-strings appear to universally show up in mirrored order following the head noun or verb make such a strongly antisymmetric approach seem problematic. Creating inverse order specifier structures via roll-up or similar mechanisms dubiously takes lack of symmetry as the starting point and symmetry as the explicandum. As noted by Abels and Neeleman (2012) the symmetries in question fall out without any specific addition if, in our terms, we simply allow the specifier and its sister to be in either order with respect to each other.

I therefore continue not to adopt the strong version of antisymmetry, where specifiers of c-strings always precede their sisters. This means that I must reject the strong immediate precedence condition on syntactic concatenation junctures that would force specifiers to precede. This however loses again the account of why a reconcatenated c-string must precede its OCS in the spellout string.

The idea that the immediate precedence relations in p-strings may be taken to express also strong immediate precedence that does not allow any intervener, alternatively to the weaker type of immediate precedence characterized in (2) above, suggests a solution. Since, given the one-dimensional linearization requirement of spellout, we cannot interpret all immediate precedence relations in the strong sense, the minimal assumption is that immediate precedence relations are in principle optionally taken to be immediate precedence in the strong sense. This will be possible for only one of two sisters to allow linearization, and there will be parametric variation as to which one. Such parametrization will determine if sister nodes are in a specifier first or a complement first structure.

Now consider reconcatenated structures again. Here the starting node C of the reconcatenated c-string is immediately preceded by P, the RCS, a member of the p-string p that contains the OCS of the c-string. Suppose that the immediate precedence relation between P and C is interpreted as in (2): P precedes C and nothing may intervene that precedes P. But this does not add anything new to the information that was present in p. If syntactic structures provide a set of precedence statements for the interpretive moduls, this additional precedence requirement between P and C will not change this set, it will remain the same as it was when reconcatenation did not take place.

Therefore if "movement" is to have any interpretive effects, the option of taking the precedence relation between P and C to be strong immediate precedence is forced. But then the c-string segment starting with C will necessarily precede its sister that precedes the OCS of the c-string. In other words the precedence condition on movement falls out.

In sum I have argued that the core properties of movement, c-command and precedence, are direct consequences of the general framework assumed here. These now follow without any movement-specific stipulations and without assuming strong antisymmetry, -- essentially from the general restrictive assumption that syntax generates one-dimensional objects.

References:

Abels, Klaus, and Ad Neeleman. 2012. Linear Asymmetries and the LCA. *Syntax* 15:25–74.

Brody, Michael. 2015. One-dimensional syntax. Ms. Linguistic Institute, Hungarian Academy of Sciences. http://ling.auf.net/lingbuzz/002863

Brody, Michael. 2017.Two advantages of Precedence Syntax. In: Bánréti et al (eds.), Boundaries Crossed: Studies at the Crossroads of morphosyntax, phonology, pragmatics, and semantics. http://ling.auf.net/lingbuzz/003761

Kayne, Richard. 1994. The Antisymmetry of Syntax. Cambridge, MA: MIT Press.

Starke, Michael 2010. Towards an elegant solution to language variation:
Variation reduces to the size of lexically stored trees. Ms. University of Tromso