Merge(X,Y) = {X,Y}

Chris Collins

August 2014

Abstract: This paper explores the history, properties and implications of the syntactic operation Merge(X,Y) = {X,Y}.

1. Unbundling

The history of generative syntax has been characterized by the gradual unbundling of syntactic transformations into different components (operations, conditions, structures, rules of interpretation). These components interact in complex ways to yield the observable syntactic data.

A good example of such unbundling can be found in Chomsky 1977: 110 where it was argued that "….we can eliminate from the grammar rules of comparative deletion, topicalization, clefting, object-deletion and 'tough movement,' rules for adjective and adjective-qualifier complements, and others, in favor of the general rule of *wh*-movement…". The general rule of *wh*-movment was given as follows (pg. 85): Move *wh*-phrase into COMP. Data was then accounted for in terms of the general rule of *wh*-movement, phrase structure rules, locality constraints, the strict cycle and principles of interpretation (the "predication rule").

Although the rule of *wh*-movement was simple, the explanation of the grammaticality status of particular sentences was quite complicated, due to the complex interactions of the various components. Chomsky 1977: 72 had only two general rules: Move *wh*-phrase and Move NP. Later, these were collapsed into one general rule Move-*α* (Chomsky 1980: 145).

Mechanisms to generate phrase structure have undergone a parallel process of unbundling. In Chomsky 1965, phrase structure was generated by rewrite rules. Such rewrite rules bundled three different kinds of information: (a) linear order (b) hierarchical grouping, and (c) syntactic category. Later work unbundled the information found in rewrite rules into different components. X'-theory imposed general constraints on phrase structure rules through what were called "phrase structure rule schema" (see Jackendoff 1977: 33). As Travis (1989: 264) explains: "Phrase structure rules also encoded two disparate types of relationships: dominance relations and precedence relations…These different relations are being teased apart in the GB framework. Dominance relations are restricted by X-bar theory. The ordering of non-heads with respect to one another is restricted by subcomponents of the grammar such as Case theory (Stowell 1981). The order of non-heads with respect to heads is restricted by one of the first parameters proposed, the headedness parameter…".

The clear break with the phrase structure tradition occurred when Chomsky (1995: 296) proposed the operation Merge: "One such operation is necessary on conceptual grounds alone: an operation that forms larger units out of those already constructed, call it Merge." (see Freidin and Lasnik 2011: 5, fn. 11)

However, early minimalism still had quite a bit of bundling. For example, all of Chomsky's early writings on minimalism defined movement as some combination of operations. For example, Chomsky (1995: 297) defined Attract/Move as incorporating the MLC and Last Resort. Chomsky (2001: 10, see also Chomsky 2000: 101, 138) defines Move as the combination of Agree, Merge and pied-piping. Crucially, the fact that Move was a combination of operations

underpinned the proposal that Move was preempted by Merge (explaining certain facts about the distribution of expletives).

Chomsky 2004: 110 (see also Epstein 1999: 320) made the crucial observation that there is just one operation, Merge, with two subcases: internal and external Merge. Merge(X,Y) is external Merge if X and Y are separate objects. Merge(X,Y) is internal Merge if either X contains Y or Y contains X. In effect, phrase structure rules and transformations collapsed into a single operation.

Even though the output of Merge in early minimalism was unspecified for linear order, it still bundled in syntactic category information: Merge(X,Y) = {X, {X, Y}} where X is the label of the syntactic object formed by Merge (see Chomsky 1995: 243). An immediate question that arose was how to determine the label. For movement, the relevant principle was that the target projects (pg. 259). For Merge (not collapsed with Move at that point) the question was left open: "I will suggest below that labels are uniquely determined for categories formed by the operation Move α, leaving the question open for Merge,…" (pg. 244).

Collins (2002) took the final step and proposed (1), the simplest possible formulation of Merge:

(1) Merge(X,Y) = {X,Y}

The main motivation for this simplification was given as follows: "The question is whether the result of the operation Merge(V, X) is {V, X} or {V, {V, X}}. The fact that Merge(V, X) combines two elements into a larger phrase is a necessary part of any theory of human language. The assumption that {V, {V, X}} is formed rather than {V, X} (that a label is chosen), goes way beyond what is necessary for grammar to make 'infinite use of finite means.'" (pg. 43). Collins (2002) then showed how various generalizations of X'-theory, subcategorization and the Minimal Link Condition would work in a label-free theory. However, he did not touch upon the issue of word order, to which I shall return in section 5 below.

Seely (2006) gave a completely different set of arguments for eliminating labels from the output of Merge. Perhaps the most compelling reason was that given the definition of c-command assumed by Seely, "…labels cannot participate in syntactic operations; they are syntactically inert." (pg. 189) So even if there were labels, they could not play any role in syntactic derivations. Seely (2006: 195, 206) also raises a number of difficult questions that labels (as part of the syntactic object formed by Merge) raise: In {see, {see, Mary}}, does the label *see* have the same selectional properties as the *see* which is the sister of *Mary*? Can the label undergo head movement? Why is the label never pronounced? All of these questions dissolve under simplest Merge.

The equation Merge(X,Y) = {X,Y} can be seen as the endpoint in the unbundling of syntactic transformations and phrase structure rules. The operation Merge gives rise to hierarchical syntactic structure. It contains no information about linear order or syntactic category. Nor does it contain any information about when syntactic transformations must apply and when they are blocked from applying. The equation Merge(X,Y) = {X,Y} seems to be the ultimate destination, in that no other simplifications are imaginable.

Furthermore, adopting Merge(X,Y) = {X,Y} can serve as a guide to the development of other parts of the theory. Working backwards from (1), it should now be possible to make significant progress in formulating Transfer.

In this paper, I assume the general background of Collins and Stabler 2014a. In addition to Merge, there is an operation which takes syntactic objects and creates structures interpretable at the interfaces. This operation is Transfer, and it has two components: Transfer_{SM} and Transfer_{CI}. Transfer_{SM} (also known as Spell-Out or externalization) yields a structure interpretable at the SM Interface. Transfer_{CI} yields a structure interpretable at the CI Interface. A central goal of this paper is to show what the consequences of adopting Merge(X,Y) = {X,Y} are for the definition of Transfer_{SM}.

The overall goal of minimalist syntax is to formulate the simplest theory of UG. As Chomsky (2014a: 2) notes: "Naturally, one seeks the simplest account of UG. One reason is just normal science: it has long been understood that simplicity of theory is essentially the same as depth of explanation. But for language there is an extra reason: UG is the theory of the biological endowment of the language faculty, and each complication of UG therefore poses a barrier to some eventual account of the origin of language, to the extent that this can be attained."

Given this goal, consider the Strong Minimalist Thesis from Berwick and Chomsky (2011: 3) (see also Chomsky 2013: 38): "…the principles of language are determined by efficient computation and language keeps to the simplest recursive operation, Merge, designed to satisfy interface conditions in accord with independent principles of efficient computation." So according to the SMT, the only operation of UG is Merge.

This formulation raises the question of the place of Transfer in UG (see Collins and Stabler 2014 who take Transfer to be part of UG). Clearly, one needs some function that operates on the output of Merge and produces structures that are interpretable at the interfaces. A natural question to ask is just how much of Transfer needs to be stipulated as part of UG and how much follows from principles of computational efficiency and the properties of the interfaces themselves. The explicit statement of Transfer_{SM} that I give in section 5 might eventually help to resolve this issue, although I will not pursue it here (see Berwick and Chomsky 2011: 38, 40 for discussion).

2. Properties of Merge

I will now be more explicit in my treatment of Merge. First, I give a formal definition of Merge in (2):

(2) Given any two syntactic objects A, B, Merge(A,B) = {A,B}

In Collins and Stabler 2014a it was stipulated that A and B be distinct. From the point of view of the current paper, such a stipulation is an example of the bundling of stipulations into the definition of Merge. If A and B must be distinct (contra Guimarães 2000, for example), then that distinctness should follow from independent principles; it should not be stipulated as an independent part of Merge (see Collins 1997: 81 for one attempt).

According to (2), A and B are syntactic objects. The recursive definition of syntactic object is given below. I return below to the definition of lexical item in section 6.

(3) X is a syntactic object iff
   i. X is a lexical item, or
   ii. X is a set of syntactic objects.

I will now enumerate some of the properties of (2).

First, Merge is iterable. Since the output of Merge is a syntactic object, it can serve as the input to Merge. If Merge(X,Y) = Z, then it is possible for Z, as a syntactic object, to be an argument of the Merge operation. I return to iterability in section 7 when I define a syntactic derivation.

Second, Merge is binary branching, essentially following Kayne 1983. Collins 1997: 76 attempts to explain binary branching in terms of computational efficiency: "The basic idea is that phrase structure is binary because binary Merge is the smallest operation that will ensure that some structure actually gets built."

Third, Merge is commutative: Merge(X,Y) = Merge(Y,X), since {X,Y} = {Y,X} (contra Collins and Stabler 2014a). However, Merge is not associative, since Merge(X, Merge(Y,Z)) = {X, {Y,Z}}, which is not in general the same as Merge(Merge(X,Y), Z) = {{X,Y}, Z}.

Fourth, the output of Merge is unspecified for linear order. I assume, following Chomsky 1995, that order is imposed at Transfer_{SM}, in a way detailed in section 5. Of course, it would be possible to define ordered Merge: Merge(X,Y) = <X,Y>, where <X,Y> is an ordered pair. However, such a definition would constitute bundling, since two different kinds of information would be encoded in the output of Merge. Furthermore, in this case, it is natural to assume that order is imposed by Transfer_{SM} to create a structure interpretable at the SM Interface.

Fifth, as already discussed, there is no label encoded in the output of Merge (see Collins 2002 and Seely 2006 for the original arguments). An immediate consequence of this is that it is not possible to define specifier or complement, which are both defined in terms of labels (See Collins and Stabler 2014a for formal definitions).

Sixth, Merge is untriggered. In order to calculate Merge(X,Y) there is no need for any feature to be checked (e.g., subcategorization features, EPP features, etc.). See Collins and Stabler 2014a for discussion. If it turns out that feature checking must take place for certain instances of Merge(X,Y), this would have to be forced by some independent principle of computational efficiency or some independent property of the interfaces, not by the definition of Merge in (2).

Seventh, given the definition of Merge, counter-cyclic internal or external Merge is not possible. In fact, any kind of replacement of terms is impossible. This point was first made clear by Collins 1997: 84 (modifying ideas of Watanabe 1995): "However, such a replacement operation would greatly complicate Merge, and so it is to be avoided. A provision would have to be added to the effect that if Merge(α,β) = γ, where α is embedded in another constituent, α must be replaced by γ. Since this provision complicates the definition of Merge and prevents us from accounting for the cycle in terms of the independently needed LCA, there is no reason to believe replacement is possible." Similarly, late merger and tucking-in are not possible. Both would require complications in (2).

The fact that Merge cannot give rise to any kind of replacement of terms is now referred to as the No Tampering Condition (see Chomsky 2005: 11). However, there is no need to stipulate an independent No Tampering Condition. The NTC follows as a theorem from the definition of Merge, since Merge is by definition unable to replace terms.

Similar remarks hold for the Extension Condition and Inclusiveness (see Collins and Stabler 2014a, and Collins 2014 for discussion). These should be understood as theorems about the system, not actual filters or conditions that need to be stipulated as part of UG.

Given the definition of Merge, it is impossible for covert movement (QR) to be counter-cyclic. Cyclic QR is consistent with the framework of Groat and O'Neil 1996, where after

internal Merge of α, the lower occurrence of α, but not the higher occurrence, would be spelled-out. Implementing this would require modifications to (13) and (15) below that I will not pursue here for reasons of space.

Eighth, Merge is the only mechanism available for building structure. It replaces both phrase structure rules and transformations (Chomsky 2004: 110). In particular, there is no bundling in the definition of Move (= Merge + Agree), for the simple reason that there is no operation Move, there is only Merge. Even though Merge has two different cases (internal Merge and external Merge), there is just one operation.

Ninth, there is no provision for the segment/category distinction in Merge. In other words, Merge does not produce adjunction structures of the type: [$_{XP}$ YP XP] (YP is adjoined to XP). Merge cannot produce head adjunction structures either. Chomsky (2004: 177) argues that an operation of Pair-Merge is needed to account for the properties of adjunction: Pair-Merge(X, Y) = <X,Y> (X adjoined to Y). However, Pair-Merge is a completely different operation from Merge, and would have to be stipulated as an independent operation of UG (going against the SMT as defined in section 1).

Tenth, there is no operation Copy (for early discussions of this issue, see Bobaljik 1995: 47, Groat 1997: 41, contra Nunes 2004: 89). In the remainder of this paper, I will use the word "copy" to mean one of the two occurrences created by internal Merge. But even though I use the word "copy", there is no copying operation. The existence of copies in this precise sense follows from (2).

Eleventh, there is no provision for the creation of "traces" in the definition of Merge. Once again the absence of traces follows from the definition of Merge; there is no need to stipulate an independent No Tampering Condition.

Twelfth, there is no provision for indices on the copies in the definition of Merge. The fact that Merge does not introduces indices falls under the inclusiveness condition (Chomsky 1995: 225, 228). But once again, there is no need to stipulate the inclusiveness condition as part of UG. Rather, it follows as a theorem from the definition of Merge.

Thirteenth, there is no notion of Chain (as a sequence of occurrences) (contra Chomsky 1995: 43, 250, 2000: 114, 116, 2001: 39, 2008), nor is there an operation Form-Chain (contra Collins 1994 and Nunes 2004). Postulating Chains or Form-Chain would once again go way beyond the definition of Merge in (2).

The consequence of points eleven, twelve and thirteen is that Transfer$_{SM}$ and Transfer$_{CI}$ must be formulated without reference to traces, indices or chains. Thus formulating Merge as (2) provides us with a clear constraint on formulating Transfer. I will meet this challenge for Transfer$_{SM}$ in section 5. An important future challenge for minimalist syntacticians/semanticists is to write a formal definition of Transfer$_{CI}$ meeting these criteria.

3. Chomsky's Labeling Algorithm

Chomsky (2008, 2013) proposes that Merge(X,Y) = {X,Y}, and that labels are identified via one of a small number of principles, the labeling algorithm. I summarize Chomsky's (2013, 2014a, 2014b) system below, using the functional notation for labels introduced in Collins and Stabler 2014a.

(4) If SO = {H, XP} where H is a head and XP is not a head, then Label(SO) = H.

As an example of (4), if Merge(see, {the, man}) = {see, {the, man}}, then Label({see, {the, man}}) = see.

(5) If SO = {XP, YP} and neither is a head, then
   a. if XP is a lower copy, Label(SO) = Label(YP).
   b. if Label(XP) and Label(YP) share a feature F by Agree, Label(SO) = <F,F>.

An example to illustrate (5) is given in (6):

(6) Who does John see?

Here the label of (6) will be <Q,Q>, since that feature is shared by *who* and the interrogative complementizer.

An immediate consequence of (5) is that the label given by (6) is not sufficient to determine linear order at Transfer_{SM}. The label is <Q,Q>, and there is no asymmetry between *who* and the rest of the clause that would allow one to determine which comes first in linear order. So like Collins 2002 and Seely 2006, Chomsky 2013 does not address the issue of labels and linear order.

From this short overview, it is clear that Chomsky accepts the fundamental proposal of Collins 2002 and Seely 2006 that labels are not encoded in the output of Merge. However, even though labels are not so encoded, they are still identified by the labeling algorithm and the labels play a role at the interfaces.

(7) Chomsky 2013: The output of Merge is label-free. Labels are determined by a labeling algorithm, and play a role at the interfaces.

Chomsky justifies the labeling algorithm by claiming that (pg. 43): "…there is a fixed labeling algorithm LA that licenses SOs so that they can be interpreted at the interfaces…". But then the question arises as to why one needs a labeling algorithm at all. Why isn't the structure of the SO, in combination with the definition of Transfer_{SM} and Transfer_{CI} sufficient to yield the correct interface representations? Pursuing this point, I argue that given independently needed assumptions about Transfer_{SM}, a labeling algorithm (an independent operation whose results feed both Transfer_{CI} and Transfer_{SM}) would be redundant. I attempt to implement this program in section 5 below.

4. Why are Labels Needed?

The question that needs to be addressed first is what role labels play in syntactic theory. There are in principle three different areas where labels could play a role:

(8) (a) internal to the syntactic computation;
   (b) at Transfer_{CI};
   (c) at Transfer_{SM}

As for (8a), it has been suggested that Merge is constrained by c-selectional information (see Collins 2002 and Collins and Stabler 2014a for references, see Seely 2006 for much relevant

discussion of c-selection in a label-free framework). I will not discuss this any further here. If c-selectional requirements do constrain the output of Merge, it is not by virtue of the definition of Merge, as shown in section 2. Of course, all the perennial questions about the scope of c-selection, and its relation to s-selection remain.

In a series of publications, Chomsky assumes that labels are needed at Transfer_{SM} and Transfer_{CI}:

(9) "Applied to two objects α and β, Merge forms the new object K, eliminating α and β. What is K? K must be constituted somehow from the two items α and β; … The simplest object constructed from α and β is the set {α, β}, so we take K to involve at least this set, where α and β are the *constituents* of K. Does that suffice? Output conditions dictate otherwise; thus verbal and nominal elements are interpreted differently at LF and behave differently in the phonological component. K must therefore at least (and we assume at most) be of the form {γ {α, β}}, where γ identifies the type to which K belongs, indicating its relevant properties. Call γ the *label* of K." (Chomsky 1995: 243)

(10) "For a syntactic object SO to be interpreted, some information is necessary about it: what kind of object is it? Labeling is the process of providing that information. Under PSG and its offshoots, labeling is part of the process of forming a syntactic object SO. But that is no longer true when the stipulations of these systems are eliminated in the simpler Merge-based conception of UG. We assume, then, that there is a fixed labeling algorithm LA that licenses SOs so that they can be interpreted at the interfaces, operating at the phase level along with other operations." (Chomsky 2013: 43).

(11) "Since the same labeling is required at CI and for the processes of externalization (though not at SM, which has no relevant structure), it must take place at the phase level, as part of the Transfer operation." (Chomsky 2014a: 4)

What information is needed at the CI Interface? I assume hierarchical information is important: {{the, man}, {bite, {the, dog}}} clearly has a different interpretation at the CI Interface than {{the, dog}, {bite, {the, man}}}. In fact, in at least one introductory semantics textbook, semantic rules of interpretation are formulated solely in terms of hierarchical structure.

Since nobody has given a concrete formal definition of Trasnsfer_{CI}, it is difficult to evaluate the claim that labels are needed in addition to hierarchical structure. The above quotes from Chomsky suggest that for a SO to be interpreted at the CI Interface, there is a need to know what kind of SO it is. For example, if the label of an SO is a question complementizer, then one knows that SO is an interrogative CP, and that information would be useful at the CI Interface. I will leave this issue aside, and focus uniquely on Transfer_{SM} in the rest of this paper.

As for (8c), labels have traditionally played an important role in linear ordering. In Principles and Parameters, specifiers, complements and adjuncts all have different ordering properties. For example, in English, specifiers precede heads, and complements follow heads. Adjuncts have a freer word order than either specifiers or complements. These notions are defined in terms of labels (see Collins and Stabler 2014a for formal definitions). See Dobashi 2003 for a label-free approach to phonological phrasing.

5. Transfer$_{SM}$

In this section, I will first present the version of Transfer$_{SM}$ (also known as Spell-Out or externalization) found in Collins and Stabler 2014a. Then, I will show how it depends on labels. Lastly, I will present a label-free alternative of Transfer$_{SM}$.

The version of Transfer$_{SM}$ in (13) below embodies three general ideas. First, the spell-out of a lexical item is a sequence of phonological segments (Phon), where a lexical item is a triple of features: LI = <Sem, Syn, Phon>. I return to the definition of lexical items in section 6.

Second, to transfer a lower copy of an element that has been internally merged, the lower copy is simply ignored. There is no separate operation of Delete that deletes lower copies. I assume that ignoring the lower copy is the result of a principle of computational efficiency: If a syntactic object has two or more occurrences, it is only spelled out at one of them. A further question is why the lower copy, instead of the higher copy, is ignored (unless covert movement is involved, see section 2, point seven). I do not pursue this question here.

Third, specifiers precede heads, and heads precede complements. This last assumption shows the dependence of (13) on labels, since specifiers and complements are defined in terms of projections of a head.

Before presenting (13), I comment on a few technical details. First, Transfer is a two-place function, since it spells out a syntactic object SO with respect to the phase that the SO is contained in. For example, suppose the phase is Phase = {that, {John, ran}}. Then one has Transfer$_{SM}$(Phase, {John, ran}). Such a system allows extraction to the edge of a phase (and successive cyclic movement), exactly as in Chomsky 2004.

Second, the symbol "^" is the concatenation symbol. X^Y means that X is followed linearly by Y and they are adjacent. I take this to be a primitive relation of the SM Interface. Third, the notion of "lower copy" is formalized in the following definition (basically c-command):

(12) A $\in${A,B} is final in SO iff there is no C contained in (or equal to) SO such that A $\in$C, and C contains {A,B}. Otherwise, A is non-final in SO.

So an occurrence of A is final if it is not c-commanded by any other occurrence of A. Otherwise, an occurrence of A is non-final. A non-final occurrence of A is what is also called a "lower copy".

With this background, Transfer$_{SM}$ is defined as follows (slightly modified from Collins and Stabler 2014a):

(13) Transfer$_{SM}$ (First Version)
For all syntactic objects SO such that either SO=Phase or SO is contained in Phase, Transfer$_{SM}$(Phase, SO) is defined as follows:

a. If SO is a lexical item, LI = ⟨Sem, Syn, Phon⟩, then Transfer$_{SM}$(Phase, SO) = Phon;

b. If SO={X,Y} and X in SO is final in Phase but Y is not, Transfer$_{SM}$(Phase, SO) = Transfer$_{SM}$(Phase, X);

c. If SO={X,Y} and X and Y in SO are final in Phase, and if Y is the complement of X, then $Transfer_{SM}$(Phase, SO) = $Transfer_{SM}$(Phase, X)^$Transfer_{SM}$(Phase, Y).

d. If SO={X,Y} and X and Y in SO are final in Phase, and if X is the specifier of Y, then $Transfer_{SM}$(Phase, SO) = $Transfer_{SM}$(Phase, X)^$Transfer_{SM}$(Phase, Y).

No provision has been made for the linearization of adjuncts, head movement or QR in this definition. I have also not defined the notion of phase, and some care needs to be taken that defining phases does not depend on the notion of labels (e.g., the maximal projection of C or v*). I do not purse these issues here.

Clause (13a) determines how lexical items are spelled-out (including functional heads). At the SM Interface, only the phonological features of the lexical item are relevant, the others are ignored.

Clause (13b) determines the spell-out of lower copies. They are simply ignored by the algorithm. An important aspect of (13b) is that it does not make reference to indices or chains of any kind. If an occurrence Y is c-commanded by an identical occurrence in a phase, then the lower occurrence is non-final. In that situation the lower occurrence of Y is not pronounced. (13) in conjunction with the definition of "final" in (12) also accounts for remnant movement, as shown in Collins and Stabler 2014a.

An issue arises, given (13b), about the distinction between copies and repetitions. I follow the general program given in Groat 2013 (see Collins and Stabler 2014b for formalization). If X and Y are identical within a phase, they are treated as copies. If X and Y are identical, but are spelled-out in different phases, they are treated as repetitions.

The crucial clauses are (13c,d) which say that a head precedes the complement, and the specifier precedes the head. Of course, in order for clauses (13c,d) to work, one needs a definition of complement and specifier, which are defined in terms of labels. These definitions are given in Collins and Stabler 2014a.

An example from Collins and Stabler 2014a shows how the system works (note that *fall* is unaccusative):

(14) Phase = {that, {John, {will, {fall, John}}}}
$Transfer_{SM}$(Phase, {fall, John})
= $Transfer_{SM}$(Phase, fall)
= /fall/

The drawback of definition (13) is that it relies on the notion of label (since it uses the label-based notions of specifier and complement), and that goes beyond the equation in (2). Therefore, I propose a label-free definition of $Transfer_{SM}$ in (15).

The definition in (15) below is structured around the following general ideas. First, the spell-out of a lexical item is a sequence of phonological segments (Phon). Second, to transfer a lower copy of an element that has been internally merged, the lower copy is simply ignored. These two principles are identical to what is found in definition (13).

Third, in a structure SO = {X,Y} where X is a lexical item and Y is not (it is a phrase), X precedes Y. I take this principle to be a principle of computational efficiency, reflecting the

fundamental left-right asymmetry imposed by the SM Interface. The principle is: the most accessible element (identified with minimal search) is spelled-out to the left.

Fourth, in a structure SO = {X,Y} where both X and Y are complex constituents (non-heads) (and neither is a lower copy), a problem arises. There is no obvious way to linearly order X and Y. I suggest a principle where if Y dominates X, then X is linearly ordered before Y. I will call this the IM (Internal Merge) Ordering Principle (IMOP). If neither X nor Y dominates the other, the structure cannot be spelled-out.

The question is whether the IMOP can be reduced to independent principles of computational efficiency. Suppose SO = {X, Y}, where X is contained in Y. The relation X is contained in Y is a partial order with minimal elements (the lexical items). The relation X precedes Y is a total order with a smallest element (the Phon of the first lexical item in the utterance). So the IMOP trades one ordering relation ("is contained in") for another ("precedence") (much as in Kayne 1994 where asymmetric c-command maps to linear order). Since this is the simplest mapping between the two orders, by computational efficiency, it is the one used by Transfer_{SM}.

Return to SO = {X,Y} where X does not dominate Y, and Y does not dominate X (there is no internal Merge). In this case, there is no ordering relation between X and Y that can be mapped to linear precedence. One possibility would be to order X and Y according to size (e.g., total number of lexical items dominated by X). But this would involve a kind of counting, and I assume that UG does not permit this kind of operation.

I assume that Transfer_{SM} has no parameters. Therefore, as in Kayne 1994, there is no head parameter. The apparent effects of the head parameter are due to various instances of internal Merge.

Given this background, the revised label-free Transfer_{SM} is given below:

(15) Transfer_{SM} (Second and Final Version)
For all syntactic objects SO such that either SO=Phase or SO is contained in Phase, Transfer_{SM}(Phase, SO) is defined as follows:

a. If SO is a lexical item, LI = ⟨Sem, Syn, Phon⟩, then Transfer_{SM}(Phase, SO) = Phon;

b. If SO={X,Y} and X in SO is final in Phase but Y is not, Transfer_{SM}(Phase, SO) = Transfer_{SM}(Phase, X);

c. If SO={X,Y} and X and Y in SO are final in Phase, where X is a lexical item and Y is not, then Transfer_{SM}(Phase, SO) = Transfer_{SM}(Phase, X)^Transfer_{SM}(Phase, Y).

d. If SO = {X,Y} and X and Y in SO are final in Phase, where X is contained in Y, then Transfer_{SM}(Phase, SO) = Transfer_{SM}(Phase, X)^Transfer_{SM}(Phase, Y).

Clauses (15a,b) are identical to (13a,b).

Clause (15c) states that if a head and a non-head are merged, then the head precedes the non-head. Clause (15c) is the label-free version of clause (13c).

Clause (15d) states that if X has undergone internal Merge, it is linearized to the left of the constituent Y out of which it has moved. Clause (15d) replaces clause (13d).

An issue looming in the background is how, given SO = {X, Y}, $Transfer_{SM}$ or $Transfer_{CI}$ finds out whether X is contained in Y. If X has K nodes (where by "node" I mean X itself and any set or lexical item contained in X), and Y has N nodes, then checking whether X is contained in Y could take in the worst case N*K steps (one checks every node of Y to see if it is identical to X). I do not pursue this issue here.

One similarity between Chomsky's system and the one in (15) is what happens with externally merged specifiers. Suppose SO = {DP, vP} where both DP and vP are final in the phase dominating SO. In Chomsky's system, such a structure cannot be labeled. In my system, such a structure cannot be linearized. To see this consider all the clauses of (15).

(15a) is not relevant, since {DP, vP} is not a lexical item. (15b) is not relevant, since by assumption DP and vP are final in the phase. (15c) is not relevant since neither DP nor vP is a lexical item. (15d) is not relevant, since DP has been externally merged. So there is no condition in (15) that is relevant. Furthermore, without labels, it is unclear how any further condition could be added to (15) that would linearize SO = {DP, vP}. There is simply no asymmetry to latch on to. I will make the assumption that if there is a SO for which $Transfer_{SM}$ does not yield a value (not even an empty string), the derivation crashes.

A criticism of my analysis is that it involves two different ordering statements (15c) and (15d). In the best of possible worlds, these two statements would be unified. In fact, Kayne 1994 proposes a unification of the head-complement case with the specifier-head case in terms of asymmetric c-command. Basically, the LCA maps asymmetric c-command onto linear order. However, Kayne's approach rests on some problematic assumptions about phrase structure and c-command, in particular the crucial definition (3) of Kayne (1994: 16), which draws a distinction between categories and segments that is not available with simplest Merge (see section 2).

Chomsky's (2013) system blocks (16b) below:

(16) a. They thought JFK was assassinated in which Texas city?
b. *They thought [$_{\alpha}$ in which Texas city [C [JFK was assassinated]]]?
c. In which Texas city did they think that JFK was assassinated?

Sentence (16a) illustrates *wh*-in-situ in English. Since the embedded C in (16b) lacks a Q feature, there is no way that α can be labeled creating a problem at the interfaces. Sentence (16c) is acceptable with successive cyclic movement, since by (5a) the intermediate copy is ignored for labeling.

On my account, there would be no problem with α at the SM Interface. In particular, $Transfer_{SM}$ would linearize (16b). A natural place to look for an explanation of the ungrammaticality of (16b) in my system is at $Transfer_{CI}$. I leave open the issue here.

6. Merge and the Lexicon

The formulation of $Transfer_{SM}$ in (15) raises the question of what happens when two lexical items are merged. For example, what happens in SO = {the, man}? (15) does not provide for any way to linearize such structures, so there should be a crash at $Transfer_{SM}$.

The question is related to the issue of the definition of lexical items. Collins and Stabler 2014a define a lexical item in the following way (see Chomsky 1995: 230) (SEM-F, SYN-F and PHON-F are universal sets of semantic, syntactic and phonological features, respectively. PHON-F* is the set of sequences of segments built up from PHON-F).

(17) A *lexical item* is a triple: LI = ⟨Sem, Syn, Phon⟩, where Sem and Syn are finite sets such that Sem ⊆ SEM-F, Syn ⊆ SYN-F, and Phon ∈ PHON-F*.

A problem with this definition is that it entails a structure (the lexical item) that is built by some operation other than Merge. In other words, some mechanism combines the three sets of features, and that mechanism is not Merge. This state of affairs seems undesirable for two reasons. First, humans have an unlimited capacity to coin lexical items, just like they have an unlimited capacity to form new phrases, suggesting that lexical items are also created by Merge. Second, adding a new mechanism (to form lexical items) would increase the complexity of UG, going against the SMT as defined in section 1.

I suggest that the three sets of features in (17) are combined by Merge. Limiting syntactic features to the categorial features (v, n, a, p), I propose the following representations (Phon = /dɔg/ and Sem = DOG):

(18) a. Merge(/dɔg/, DOG) = {/dɔg/, DOG}
b. Merge(n, {/dɔg/, DOG}) = {n, {/dɔg/, DOG}}

On this view, n simply has no phonological features, so that Transfer_{SM}(Phase, n) = ∅ (the empty string). I leave aside the question of why other combinations are not possible, e.g., Merge(n, /dɔg/), etc. Given (18b), the natural assumption is that n will be linearized preceding {/dɔg/, DOG}}, because it is the most accessible element.

A natural question is to ask whether functional categories such as *the* are decomposed in the same way as *dog*. For closed class, functional categories, I will assume that there are only syntactic features (Syn), which are also semantically interpreted. So the definite article will have the following representation, where DEF is a set of syntactic features (possibly a singleton set):

(19) Merge(/ðə/, DEF) = {/ðə/, DEF}

Given these assumptions, I now define lexical items in the following way:

(20) A *lexical item* is a syntactic object of the form {Phon, Syn} or {Phon, Sem}
(that is, a pairing of Phon features with either Syn or Sem feaures).

The SO = {the, dog} will now be linearized in the following way, following the procedure layed out in (15) (changing only the definition of lexical item, and taking into account the discussion following (18)):

(21) Transfer_{SM}(Phase, {{/ðə/, DEF}, {n, {/dog/, DOG}}})
= Transfer_{SM}(Phase, {/ðə/, DEF}) ^ Transfer_{SM}(Phase, {n, {/dɔg/, DOG}})
= /ðə/ ^ Transfer_{SM}(Phase, n) ^ Transfer_{SM}(Phase, {/dɔg/, DOG})
= /ðə/ ^ ∅ ^ /dɔg/
= /ðə/ ^ /dɔg/

In order for Merge to operate as in (18) and (19) one of the two definitions in (2) and (3) needs to change. Consider (2). A sequence of phonological segments Phon is not a syntactic object, so Merge(Phon, Syn) would be undefined according to (2). One possibility is to redefine (2) to allow merger of more than just syntactic objects. Another possibility is to redefine (3) so that Phon counts as a syntactic object.

The output of Merge in (18a) is similar to what is called a "root" in Distributed Morphology (Embick and Noyer 2007: 295). However, in my system, there is no primitive notion of "root". Furthermore, there is no notion of Vocabulary Insertion (Embick and Noyer 2007: 297). The rule of Vocabulary Insertion is quite complicated necessitating many auxiliary assumptions (e.g., a list of vocabulary items, the placeholder Q for the phonological exponent, conditions on insertion, etc), making it unclear how Vocabulary Insertion would fit into the SMT (see section 1). Similarly, in the system of this paper (based on simplest Merge), there is no possibility of post-syntactic insertion of ornamental morphemes (agreement, case, theme vowels, see Embick and Noyer 2007: 305) or post-syntactic lowering (Embick and Noyer 2007: 319). There is only Merge, and all "morphology" must ultimately be explained in terms of Merge.

To take a simple example, Embick and Noyer (2007: 298,299) claim that there are two vocabulary items, one for the regular plural (z ↔ [pl]) and one for the irregular plural ([pl] ↔ -en in the context of the roots OX and CHILD). These two vocabulary items are in competition to be inserted at the same terminal node specified with the abstract morpheme [pl]. But for me, both *oxen* and *oxes* (and even *ox*) are possible plural forms, so there can be no competition. Furthermore, for me there is a clear difference between *oxens* (marginal) and *oxesen* (completely ungrammatical), which suggests that the regular plural and *–en* do not even occupy the same position in the DP. These facts together suggest that there are two different morphemes {pl1, z} and {pl2, en} each of which can be merged to a noun.

Similar remarks can be made about the CI side. Operations which produce interpretations that mimic the results of Merge should be avoided (e.g., implicit arguments that are present semantically but not syntactically, quantifiers or operators present in the semantics but not the syntax, or type shifting rules). I do not pursue these issues for reasons of space.

7. Derivations

It is not possible to study Merge without specifying how it enters into derivations. As noted in section 2, the output of Merge is a syntactic object, and hence the output of Merge can also be one of the input arguments to Merge. This makes it possible to define a derivation in simple terms. Because of point seven, in section 2, derivations will be bottom-up, creating larger and larger structures as the derivation progresses.

(22) A *derivation* D from lexicon L is a finite sequence of steps such that each step is one of the following:

i. A lexical item

ii. Merge(X,Y) = {X,Y}, where both X and Y are syntactic objects that appear earlier in the derivation (either as lexical items or as the output of Merge).

Even though derivations have a finite length, there are an unlimited number of derivations, allowing for an account of the fact that language makes "infinite use of finite means".

There is no need for a numeration or a lexical array. There is also no need for an operation Select (see Collins and Stabler 2014a), although listing the lexical item as its own line of the derivation (as in (23b) below) could be viewed as selection of a lexical item out of the lexicon.

For example, given the definition in (22), the following is a derivation:

(23) a. see

b. John

c. Merge(see, John) = {see, John}

The definition in (22) contrasts with the usual definition of a derivation found in minimalist literature as a sequence of workspaces, where each workspace is a set of syntactic objects. On the sequence of workspaces definition, Merge(X,Y) creates the syntactic object {X,Y} and adds it to the workspace. Crucially, the two syntactic objects X and Y are removed from the workspace (Chomsky 1995: 226, 243, see Collins and Stabler 2014a for discussion and references). As Collins and Stabler 2014a have shown, implementing the sequence-of-workspaces definition requires a great detail of stipulation and complexity. Hence, the simpler definition in (22), not involving a sequence of workspaces, should be preferred on minimalist grounds (see the discussion of the SMT in section 1).

One issue with (22) is the status of Transfer in the derivation. There are two possibilities. One is to view Transfer as an operation that happens automatically when a strong phase has been built. On this view, there is no reason to modify (22). Transfer simply occurs when the conditions are met.

The second (more common) way is to view Transfer as an operation ordered amongst Merge operations in a derivation. On this view, the definition in (22) must be altered to include Transfer:

(24) A *derivation* D from lexicon L is a finite sequence of steps, such that each step is one of the following:

i. A lexical item

ii. Merge(X,Y) = {X,Y}, where both X and Y are syntactic objects that appear earlier in the derivation (either as lexical items or as the output of Merge).

iii. Transfer_{SM}(Phase, SO) = <PHON, SEM>, where Phase is a syntactic object that appears earlier in the derivation, and SO is contained in Phase.

I do not tackle the Assembly Problem here. That is, I do not show how the outputs of Transfer_{SM} throughout the derivation are assembled correctly. See Collins and Stabler 2014a for

extensive discussion of this issue. I also do not tackle the definition of phases in a label-free system.

These simple definitions of a derivation have far reaching implications for syntactic analysis, including the issue of two peak structures (originally studied by Collins 1997: 83, Groat 1997: 101-104 and more recently by Epstein, Kitahara and Seely 2012), sideward movement and head movement (see in particular Bobaljik and Brown 1997). I will not pursue these issues for reasons of space.

8. Agree

Merge(X,Y) = {X,Y} is indispensable as a syntactic operation. It is what allows the language faculty to generate and parse an unlimited number of expressions, given the finite means of the human mind/brain. The null hypothesis is that there are no other operations. An additional operation OP would go against the SMT as defined in section 1.

Consider Agree from this perspective. Agree creates dependencies that are similar to the dependencies created by internal Merge. With Agree, some feature set (e.g., 3SG) has two different occurrences, exactly like a syntactic object that has undergone internal Merge. Given this background, I propose (25) (see Seely 2014 for a similar conclusion from a different perspective):

(25) There is no operation Agree in UG

The effects of Agree need to be captured by Merge. For example, subject-verb agreement could be analyzed as follows. I make the simplifying assumption that the phi-features of the DP are located on D (written *the[phi]*). It would be more accurate to say that the phi-features of the DP are distributed throughout the DP (e.g., the number feature is found in the Num head).

(26) "The man falls."
- a. the[phi]
- b. man
- c. Merge(the[phi], man)
- d. fall
- e. Merge(fall, {the, man}) = {fall, {the, man}}
- f. T
- g. Merge(phi, T) = {phi, T}
- h. Merge({phi,T}, {fall, {the man}}) = {{phi, T}, {fall, {the, man}}}
- i. Merge({the, man}, {{phi,T}, {fall, {the man}}})
  = {{the, man}, {{phi,T}, {fall, {the man}}}}

The SO = {the, man} has phi-features. These phi-features are also merged with T, creating a dependency. This instance of Merge is neither internal nor external Merge, but rather sidewards Merge (as in the Bobaljik and Brown 1997 analysis of head movement).

Support for this way of looking at agreement concerns the No Tampering Condition. Suppose that T has unvalued phi-features (uPhi) that are valued. Suppose furthermore that T is contained in SO. Then after valuation, T has changed and SO has changed. Agree has created new syntactic objects, which we can call T' and SO'. This is a violation of the NTC, as defined

formally in Collins and Stabler 2014a. Of course, it is possible to avoid the conclusion that Agree violates the NTC if one changes the definition of the NTC. Alternatively, one might want to limit the scope of the NTC to Merge, claiming that Agree is part of Transfer and that Transfer is not subject to the NTC. However, all these issues simply evaporate if Agree is not part of UG, and clearly the derivation in (26) does not violate any version of the NTC.

In order for Merge to operate as in (26) one of the two definitions in (2) and (3) needs to change. Consider (2). Phi is not a syntactic object, so Merge(Phi, T) would be undefined according to (2). One possibility is to redefine (2) to allow merger of Phi. Another possibility is to redefine (3) so that Phi is counted as a syntactic object. The issues are similar to those raised in section 6.

This approach to agreement leaves open many questions. For example, how are the two occurrences of Phi interpreted by Transfer_{SM} and Transfer_{CI}? Also, what triggers agreement in the first place? I will not address these issues here for reasons of space.

8. Conclusion

In this paper, I have explored the history, properties and implications of the syntactic operation Merge(X,Y) = {X,Y}.

I sketched the historical development of the equation Merge(X,Y) = {X,Y}, claiming that it is the final destination in a process of unbundling that has taken place since the beginning of generative grammar. Then, I presented thirteen properties of Merge.

I discussed the consequences of simplest Merge for the definition of Transfer_{SM} (leaving open the status of Transfer_{CI}). I showed how the definition of Transfer_{SM} can be modified to linearize a syntactic structure without reference to labels.

I discussed the consequences of simplest Merge for the nature of the lexicon, where I proposed that lexical items are created by Merge. I argued that there is no operation Agree in UG, and showed how Merge fits into a simple (perhaps the simplest) definition of a derivation.

Acknowledgements:
I would like to thank Yoshi Dobashi, Erich Groat and Daniel Seely for comments on an earlier version of this paper.

References:

Berwick, Robert and Noam Chomsky. 2011. The Biolinguistic Program: The Current State of its Development. In Anna Maria Di Sciullo and Cedric Boeckx (eds.), *The Biolinguistic Enterprise*. Oxford: Oxford University Press.

Bobaljik, Jonathan. 1995. In Terms of Merge. In Rob Pensalfini and Hiroyuki Ura (eds.), *Papers on Minimalist Syntax,* pgs. 41-64. Cambridge: MITWPL.

Bobaljik, Jonathan and Samuel Brown. 1997. Head Movement and the Extension Requirement. *Linguistic Inquiry* 28.2, pgs. 345-356.

Chomsky, Noam. 1965. *Aspects of the Theory of Syntax*. Cambridge: MIT Press.

Chomsky, Noam. 1977. On Wh-Movement. In Peter Culicover, Thomas Wasow and Adrian Akmajian (eds.), *Formal Syntax*, pgs. 71 – 132. Academic Press, New York.

Chomsky, Noam. 1980. *Rules and Representations*. New York: Columbia University Press.

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.

Chomsky, Noam. 2000. Minimalist Inquiries. In *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik*, eds. Roger Martin, David Michaels, Juan Uriagereka (eds.), pgs. 89-155. Cambridge, MA: MIT Press.

Chomsky, Noam. 2001. Derivation by Phase. In *Ken Hale: a Life in Language*, ed. Michael Kenstowicz, pgs. 1-52. Cambridge, MA: MIT Press.

Chomsky, Noam. 2004. Beyond Explanatory Adequacy. In *Structures and Beyond*, ed. Adriana Belletti, pgs. 104-131 (originally published as: Chomsky, Noam. 2001. Beyond Explanatory Adequacy. *MIT Occasional Papers in Linguistics 20. MIT Working Papers in Linguistics*.)

Chomsky, Noam. 2005. Three factors in language design. *Linguistic Inquiry* 36, pgs. 1-22.

Chomsky, Noam. 2008. On Phases. In *Foundational Issues in Linguistic Theory*, eds. Robert Freidin, Carlos P. Otero, and Maria Luisa Zubizarreta, pgs. 133-166. Cambridge, MA: MIT press.

Chomsky, Noam. 2013. Problems of Projection. *Lingua* 130, pgs. 33-49

Chomsky, Noam. 2014a. Problems of Projection: Extensions. Ms., MIT.

Chomsky, Noam. 2014b. Lectures on syntax at MIT. http://whamit.mit.edu/2014/06/03/recent-linguistics-talks-by-chomsky/

Collins, Chris. 1994. Economy of Derivation and the Generalized Proper Binding Condition. *Linguistic Inquiry* 25, pgs. 45-61.

Collins, Chris. 1997. *Local Economy*. Cambridge, MA: MIT Press.

Collins, Chris. 2002. Eliminating Labels. In *Derivation and Explanation in the Minimalist Program*, eds. Samuel David Epstein and T. Daniel Seely, pgs. 42-64. Oxford: Blackwell.

Collins, Chris. 2014. Why Formalize? Blog Post, Faculty of Language. February 10, 2014. http://facultyoflanguage.blogspot.com/2014/02/where-norbert-posts-chriss-revised-post.html

Collins, Chris and Edward Stabler. 2014a. A Formalization of Minimalist Syntax. Ms., NYU and UCLA.

Collins, Collins and Edward. Stabler. 2014b. A Formalization of a Phase-Based Approach to Copies versus Repetitions. Ms., NYU and UCLA.

Dobashi, Yoshihito. 2003. *Phonological Phrasing and Syntactic Derivation.* Doctoral dissertation, Cornell University.

Embick, David and Rolf Noyer. 2007. Distributed Morphology and the Syntax-Morphology Interface. Gillian Ramchand and Charles Reiss (eds.), *The Oxford Handbook of Linguistic Interfaces*. Oxford University Press, Oxford.

Epstein, Samuel D. 1999. Un-Principled Syntax: The Derivation of Syntactic Relations. In *Working Minimalism*, eds. Samuel David Epstein and Norbert Hornstein, pgs. 317-145. Cambridge, MA: MIT Press.

Epstein, Samuel D., Hisatsugu Kitahara and Seely, T. Daniel. 2012. Structure Building that Can't Be. In *Ways of Structure Building*, ed. M. Uribe-Etxebarria and V. Valmala, pgs. 253-270. Cambridge: Cambridge University Press.

Freidin, Robert and Howard Lasnik. 2011. Some Roots of Minimalism in Generative Grammar. In Cedric Boeckx (ed.), *Linguistic Minimalism*, pgs. 1-26.Oxford: Oxford University Press.

Groat, Erich. 1997. A Derivational Program for Syntactic Theory. Doctoral Dissertation, Harvard.

Groat, Erich. 2013. Is Movement Part of Narrow Syntax? Handout, Goethe University Frankfurt, May 22, 2013.

Groat, Erich and John O'Neil. 1996. Spell-Out at the LF Interface. In Werner Abraham, Samuel David Epstein, Höskuldur Thráinsson and C. Jan-Wouter Zwart (eds.), *Minimal Ideas*, pgs. 113-139. Amsterdam: John Benjamins.

Guimarães, Max. 2000. In Defense of Vacuous Projections in Bare Phrase Structure. In *University of Maryland Working Papers in Linguistics 9*, eds. G. Maximiliano, L. Meroni, C. Rodrigues and I. San Martin, pgs. 90-115.

Jackendoff, Ray. 1977. *X'-Syntax: A Study of Phrase Structure*. Cambridge, MA: MIT Press.

Kayne, Richard. 1983. *Connectedness and Binary Branching.* Dordrecht: Foris Publications.

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

Nunes, Jairo. 2004. *Linearization of Chains and Sideward Movement*. Cambridge, MA: MIT Press. MIT Press,

Seely, T. Daniel. 2006. Merge, Derivational C-Command, and Subcategorization in a Label-Free Syntax. In *Minimalist Essays*, ed. Cedric Boeckx. Amsterdam: John Benjamins.

Seely, T. Daniel. 2014. Agreement is the Shared Prominent Feature Option of POPs Labeling Algorithm. Ms., Eastern Michigan University.

Stowell, Timothy. 1981. Origins of Phrase Structure. Doctoral Dissertation, MIT.

Travis, Lisa. 1989. Parameters of Phrase Structure. In Mark Baltin and Anthon Kroch (eds.), *Alternative Conceptions of Phrase Structure.* Chicago: Chicago University Press.

Watanabe, Akira. 1995. Conceptual Basis of Cyclicity. In Rob Pensalfini and Hiroyuki Ura (eds.), *Papers on Minimalist Syntax*, pgs. 269-291. Cambridge: MITWPL.