# A Formalization of Minimalist Syntax

**Article** *in* Syntax · March 2016

DOI: 10.1111/synt.12117

CITATIONS
90

READS
1,880

**2 authors:**

Chris Collins
New York University
**99** PUBLICATIONS **1,996** CITATIONS
SEE PROFILE

Edward P. Stabler
University of California, Los Angeles
**120** PUBLICATIONS **1,992** CITATIONS
SEE PROFILE

Syntax
A Journal of Theoretical, Experimental and Interdisciplinary Research

*Syntax* 19:1, March 2016, 43–78 DOI: 10.1111/synt.12117

# A Formalization of Minimalist Syntax

*Chris Collins and Edward Stabler*

*Abstract.* The goal of this paper is to give a precise, formal account of certain fundamental notions in minimalist syntax. Particular attention is given to the comparison of token-based (multidominance) and chain-based perspectives on Merge. After considering a version of Transfer that violates the No-Tampering Condition (NTC), we sketch an alternative, NTC-compliant version.

## 1. Introduction

The goal of this paper is to give a precise, formal account of certain fundamental notions in minimalist syntax, including Merge, Select, Transfer, occurrences, workspace, labels, and convergence. Many issues are not treated for reasons of space, including head movement, Pair-Merge (adjunction), Quantifier Raising, Agree, locality conditions, feature inheritance, and so forth. We hope that the framework in this paper will be a useful first step toward these topics. We would like this formalization to be useful as a toolkit for minimalist syntacticians in formulating new proposals and evaluating their own proposals, both conceptually and empirically.

Our basic approach bears a resemblance to formal proposals of Gärtner (2002) and other foundational studies,[1] but we stick to simpler (and more mainstream) structures and operations. Less directly, this work is similar to the minimalist grammars devised by Stabler (1997) and the work that it has given rise to.[2] Those grammars were simplified to facilitate computational assessment, but here we make an effort to stay close to mainstream formulations. The goal here is to precisely understand and assess some of the most fundamental grammatical proposals.

We use basic set theory to represent syntactic objects, with standard notation: $\in$ (is an element of), $\cup$ (set union), $\subseteq$ (is a subset of), and $\subset$ (is a proper subset of). The empty set is written $\{\}$ or $\varnothing$. Given sets S and T, the difference S - T = $\{x|x \in S, x \notin T\}$. Angle brackets are used for sequences $\langle S_1,S_2,\ldots,S_n\rangle$, and when $n = 0$, the empty sequence is written $\langle\rangle$ or $\varepsilon$. We call sequences of length 2 "pairs," and sequences of length 6 "6-tuples." As usual, free variables in definitions are understood to be universally quantified. For example, "W is a workspace iff. . ." means "For all W, W is a workspace iff.. . ."

We would like to thank Noam Chomsky, Yoshi Dobashi, Thomas Graf, Erich Groat, Kevin Guzzo, Greg Kobele, Terje Lohndal, Jens Michaelis, Yohei Oseki, Paul Postal, T. Daniel Seely, and Vera Zu for comments on the ideas found in this paper. We presented this work at the European Summer School in Logic, Language, and Information (July 2009), the MIT Syntax–Semantics Reading Group (September 2009), the Fifth International Conference on Formal Linguistics (Guangzhou, China, December 2011), and a seminar on syntax at New York University (Spring 2014).

[1] See also the foundational studies of Veenstra (1998), Kracht (1999, 2001, 2008), and Frampton (2004). More detailed comparisons will be made at a later point in the paper.

[2] See Michaelis 2001, Harkema 2001, Kobele 2006, Stabler 2010, Hunter 2011, Salvati 2011, and Graf 2013.

## 2. Preliminary Definitions

**Definition 1**

Universal Grammar is a 6-tuple: ⟨PHON-F,SYN-F,SEM-F,Select,Merge,Transfer⟩.

PHON-F, SYN-F, and SEM-F are universal sets of features. Select, Merge, and Transfer are universal operations. Select is an operation that introduces lexical items into the derivation. Merge is an operation that takes two syntactic objects and combines them into a syntactic object. Transfer is an operation that maps the syntactic objects built by Merge to pairs ⟨PHON,SEM⟩ that are interpretable at the interfaces. Select, Merge, and Transfer are defined later in the paper. Another operation to add to Definition 1 is Agree, but for reasons of space, we do not formalize Agree here. Definition 1 is intended to capture what is invariant in the human language faculty.

We assume that Universal Grammar (UG) specifies three sets of features (Chomsky 1995:230, 2000b:170): semantic features (SEM-F; Chomsky 2000b:120), phonetic features (PHON-F), and syntactic features (SYN-F). SEM-F may include features pertaining to aktionsart, thematic roles, negation, focus, topic, tense, aspect, quantification, definiteness, plurality, causation, and so forth. PHON-F may contain phonological features such as [+voiced] or [+ATR]. PHON* is the set of strings of segments, each of whose features are chosen from PHON-F. SYN-F includes features that play a role in the syntactic derivation. SYN-F may include syntactic categories such as N, V, P, and Adj but also subcategorization features, Extended Projection Principle (EPP) features, Case features, and unvalued features [uF] (e.g., ϕ-features of T and perhaps pronouns; see Collins & Postal 2012). We call the features of SYN-F that are satisfied by Merge (i.e., subcategorization features and EPP) trigger features (see sect. 7).

As noted earlier, we do not focus here on the theory of features. For our present purposes, we do not need to be clearer about what the features are, but we assume that they are basic elements, different from the sorts of syntactic objects we will derive in the syntax.

**Definition 2**

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*.

For some lexical items, SEM, SYN, or PHON could be empty. We have simplified greatly at this point, putting aside many interesting questions (e.g., What is the relation between SEM and SYN for a particular lexical item? How are the feature sets in SEM and SYN structured?). For more on features in minimalist syntax, see Adger 2010.

**Definition 3**

A *lexicon* is a finite set of lexical items.

**Definition 4**

An I-language is a pair ⟨Lex,UG⟩ where Lex is a lexicon and UG is Universal Grammar.

In order to allow structures in which a given lexical item occurs twice in a structure, the lexical items in our structures are indexed with integers (which is basically equivalent to Chomsky 1995:227,[3] contra Kitahara 2000). For example, in the sentence *The dog saw the other dog*, there are two tokens of the single lexical item *dog*, tokens with different numerical indices. The integer in the lexical item token plays no other role in the syntactic computation. For example, the integer will not be used in "counting." In fact, any other infinite set of distinguishing marks would serve just as well (e.g., ⟨dog,!⟩, ⟨dog,!!⟩, ⟨dog,!!!⟩, etc.; see Groat 2013 for critical discussion).

As discussed later (see Theorem 6), when lexical items are indexed, there is no need for additional coindexing of elements related by movement, because the lexical indices suffice to unambiguously represent the results of movement relations. We do not take up the question of how lexical indices relate to the "referential indices" of the binding theory. The referential indices of binding theory play no role in this paper.

**Definition 5**

A *lexical item token* is a pair ⟨LI,$k$⟩ where LI is a lexical item and $k$ is an integer.

When context makes our intentions clear, we will use LI to mean either lexical item or lexical item token. For convenience, when the integer of a lexical item token is indicated, we usually write it as a subscript: ⟨John,$k$⟩ = John$_k$, where *John* is itself an abbreviation for a triple of SEM, SYN, and PHON features.

A lexical array is a set of lexical item tokens with distinct indices:

**Definition 6**

A *lexical array* (LA) is a finite set of lexical item tokens.

A lexical array LA could contain two tokens of *dog*, for example *dog*$_7$ and *dog*$_{43}$. These two tokens are distinct, both for syntactic operations and at the interfaces.

Given a lexical array with tokens explicitly marked in this way, we do not need an additional notion of *numeration*. Chomsky (1995:225) defines a numeration as "a set of pairs (LI,$i$), where LI is an item of the lexicon and $i$ is its index, understood to be the number of times that LI is selected." For example, if the pair (dog,2) is in the numeration, *dog* will be selected twice in the derivation, as in the sentence *The dog*

[3] "But the syntactic objects formed by distinct applications of Select to LI must be distinguished; two occurrences of the pronoun *he*, for example, may have entirely different properties at [Logical Form]. $l$ and $l'$ are thus marked as distinct for $C_{HL}$ if they are formed by distinct applications of Select accessing the same lexical item of N." In this quote, N is the numeration. In our formalization, the items are always already distinct in the "lexical array"; see Definition 6. A chain-based alternative is considered in section 6.

*sees the other dog*. Our notion of lexical item token also allows the lexical item *dog* to be selected twice (once when the token $dog_7$ is selected and once when the token $dog_{43}$ is selected). Hence there is no need for a numeration.

**Definition 7**

X is a *syntactic object* iff

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

The Merge function, defined in section 2, maps syntactic objects to syntactic objects. Chomsky (1995:243) offers a definition of syntactic object that incorporates the notion of *label*, but we separate the definition of syntactic objects and introduce labels later (see sect. 7).

We also need some definitions to refer to the relations between syntactic objects (cf. Chomsky 2000a:116, in which "contain" is taken to be reflexive).

**Definition 8**

Let A and B be syntactic objects, then B *immediately contains* A iff $A \in B$.

**Definition 9**

Let A and B be syntactic objects, then B *contains* A iff

(i) B immediately contains A, or
(ii) for some syntactic object C, B immediately contains C and C contains A.

Notice that with these definitions, "immediately contains" is the "has as a member" relation, and "contains" is the transitive closure of that relation. That is, if A contains B, then B is a member of A, or a member of a member of A, and so on. Definitions 8 and 9 also cover the case of workspaces, so that one may talk of a workspace W containing a syntactic object A.

## 3. Workspaces, Select, and Merge

A derivation of a syntactic object is a series of stages that constructs a single syntactic object from some lexical item tokens. Each stage in the derivation is defined by a lexical array and a workspace (cf. Chomsky 1995:226, 2013:41; Collins 1997:3; Epstein, Kitahara & Seely 2012:254, 269):

**Definition 10**

A *stage* is a pair $S = \langle LA,W \rangle$, where LA is a lexical array and W is a set of syntactic objects. We call W the *workspace* of S.

A derivation will be defined later (Definition 14) as a sequence of stages meeting certain requirements. The lexical array includes all the lexical item tokens that may be

introduced into a particular derivation at a particular stage. A workspace includes all the syntactic objects that have been built up at a particular stage in the derivation. Note that by Definition 7, a workspace is a syntactic object. However, by convention, we reserve the term "syntactic object" for those elements built up in the course of the derivation and contained in the workspace.

In minimalist literature, the term "workspace" is also used in a sense where two syntactic objects that are being built in parallel occupy two different workspaces. These two different workspaces are combined at some point in the derivation (see Nunes 2004:140). We do not use the term "workspace" in this sense, although formalizing this alternative in our framework would not be difficult.

**Definition 11**

For any syntactic object X and any stage $S = \langle LA,W \rangle$ with workspace W, if $X \in W$, X is a *root* in W.

The Merge operation is constrained to apply to a root (see the definition of derivation in Definition 14), and the operation acts at the root in the sense that it embeds a root into a more complex syntactic object.

Before defining the operation Merge, we first define an operation that selects a lexical item token from the lexical array in a stage:

**Definition 12**

Let S be a stage in a derivation $S = \langle LA,W \rangle$.
If lexical token $A \in LA$, then $Select(A,S) = \langle LA - \{A\}, W \cup \{A\} \rangle$

Select is an operation that takes a lexical item token from the lexical array and places it in the workspace, at which point it is a root, available to be merged. Note that Select is only defined on stages that contain nonempty lexical arrays.[4]

Merge is defined on syntactic objects. It is a function that maps pairs of syntactic objects to new syntactic objects in the following simple way, as argued for in Collins 2002 (see also Chomsky 1995:243, 2007:8):

**Definition 13**

Given any two distinct syntactic objects A, B, $Merge(A,B) = \{A,B\}$.

Merge takes two syntactic objects and combines them into a single syntactic object. According to Definition 7, lexical item tokens are syntactic objects, so Merge can combine them. This is the basic structure building operation of syntax.

[4] Collins (1997:89–90) and Frampton & Gutmann (2002:93) argue against the existence of a numeration/lexical array. On such a theory, a stage in a derivation is simply defined as a workspace. The operation Select would then need to introduce a lexical item token directly into the workspace: $Select(LI,W) = W \cup \{LI\}$.

The distinctness clause means that no syntactic object A can be merged with itself. In other words, Merge(A,A) is undefined. Ultimately, the distinctness clause implements the lack of "nonbranching projections." For example, Merge(A,A) = {A,A}, and by set theory, {A,A} = {A}. Therefore, Merge(A,A) = {A}. So ruling out Merge(A,A) prevents {A}, which is nonbranching, from being generated.[5]

We make no distinction between external Merge and internal Merge.[6] They are not two separate operations (see, e.g., Chomsky 2005:12). Rather, external Merge corresponds to the case of Merge where A, B $\in$ W (a workspace). Internal Merge corresponds to the case of Merge where A $\in$ W, and A contains B. In the latter case, A is called the "target of movement."

Consider the following example of Merge. Let $see_1$ and $John_1$ be lexical item tokens in some workspace W, then:

(1) $\text{Merge}(\text{see}_1,\text{John}_1) = \{\text{see}_1,\text{John}_1\}$

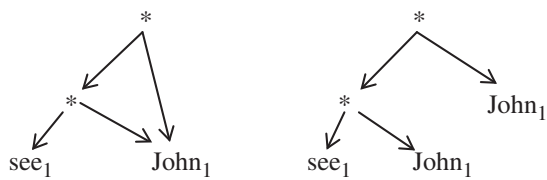
The result of this operation, $\{\text{see}_1,\text{John}_1\}$, has no syntactic category label (e.g., VP). The formalism for labels will be developed at a later point in this paper, after the operation of "triggered merge" is defined.

Now suppose that the workspace W is $\{\{\text{John}_1,\text{see}_1\}\}$, then the following Merge operation is also possible:

(2) $\text{Merge}(\{\text{John}_1,\text{see}_1\},\text{John}_1) = \{\text{John}_1,\{\text{John}_1,\text{see}_1\}\}$

When the second argument of Merge, the syntactic object $John_1$, occurs inside the first argument, it appears in two places in the result (a multidominance structure). This falls under the internal Merge subcase of Merge. Graph-theoretically, either of the two following representations are possible (see Gärtner 2002:sect. 3.3.2 on the fact that sets allow for alternative ways of being pictured by graphs).

(3)




In this kind of "set membership" diagram, the internal nodes labeled * are sets, syntactic objects, with arcs pointing to their elements. We regard the two representations in (3) as equivalent. In both cases, there are two arcs pointing to

[5] Guimarães (2000) argues that Merge(A,A) ("Self-Merge") is possible. His main motivation is to show how a version of Bare Phrase Structure theory allowing Self-Merge is consistent with the Linear Correspondence Axiom (LCA). However, we note that our linearization algorithm (see Definition 41 for $\text{Transfer}_{\text{PF}}$), which is based on the LCA, has no need of nonbranching projections.

[6] Compare early minimalism, where there is a distinction between GT (generalized transformation) and Move $\alpha$ (Chomsky 1995:189). Similar distinctions existed in earlier frameworks (e.g., phrase-structure rules versus transformations).

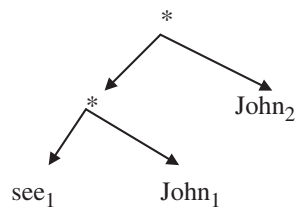
the syntactic object $John_1$. In both cases, a single syntactic object $John_1$ has two *occurrences* in a sense to be defined later.

For comparison, consider the following example of Merge, given a workspace W = {{John$_1$,see$_1$},John$_2$}:

(4) Merge({John$_1$,see$_1$},John$_2$) = {{John$_1$,see$_1$},John$_2$}

In this structure, two distinct lexical item tokens correspond to the lexical item *John*. These lexical item tokens appear in two different positions in the structure. Graph-theoretically, it can be represented as in (5).

(5)




We can now define a derivation as a sequence of stages, where a stage includes a workspace and a lexical array.

**Definition 14**

A *derivation* from lexicon L is a finite sequence of stages $\langle S_1,\ldots,S_n\rangle$, for $n \geq 1$, where each $S_i = \langle LA_i,W_i\rangle$, such that

(i) For all LI and $k$ such that $\langle LI,k\rangle \in LA_1$, LI $\in$ L,
(ii) $W_1 = \{\}$ (the empty set),
(iii) for all $i$, such that $1 \leq i < n$, either
(derive-by-Select) for some A $\in LA_i$, $\langle LA_{i+1},W_{i+1}\rangle$ = Select(A,$\langle LA_i,W_i\rangle$), or
(derive-by-Merge) $LA_i = LA_{i+1}$ and the following conditions hold for some A,B:
(a) A $\in W_i$,
(b) either A contains B or $W_i$ immediately contains B, and
(c) $W_{i+1} = (W_i - \{A,B\}) \cup \{Merge(A,B)\}$.

Condition (b) is a disjunction. The first disjunct is called *internal Merge* and the second disjunct is *external Merge*. Internal and external Merge both involve exactly the same operation. They differ only in where the argument B is found.

This definition has as a consequence that operations cannot apply in parallel, because a derivation is a sequence of stages and two consecutive stages are related by one operation. How to formalize a derivation in which operations can apply in parallel, and whether there are empirical differences between the sequential and parallel approach, are matters for future research.

In this definition of derive-by-Merge, the lexical array does not change from one stage to the next, because it is not altered by Merge (only by Select). If A, B ∈ W (the external Merge subcase), then the effect of the derive-by-Merge relation is to remove both A and B from W. If A ∈ W, and B is contained in A (the internal Merge subcase), then the effect of the derive-by-Merge relation is to remove only A from W (because B is not a member of W). In this case, we say that B undergoes internal Merge.

The condition that $A \in W_i$ in Definition 14 encodes the fact that Merge is "at the root" (Chomsky 1995:248). Various modifications of derive-by-Merge would yield other possibilities. For example, if the condition were "$A \in W_1$ and $B \in W_1$," then only external Merge would be allowed. If the condition were "$A \in W_1$ and B is contained in $W_1$," then in addition to internal and external Merge, sideward Merge would also be possible. If the condition were "A and B are contained in $W_1$," then Merge would be possible between nonroots. We have chosen the most conservative possible condition in this formalization (see Nunes 2004, Citko 2005, Riemsdijk 2006, and Wilder 2008 for empirical arguments for sideward Merge). However, it is important to highlight that relatively minor changes in our assumptions about derive-by-Merge would allow sideward Merge.

**Definition 15**

A syntactic object A is *derivable from lexicon* L iff there is a derivation $\langle\langle LA_1, W_1\rangle,\ldots,\langle LA_n,W_n\rangle\rangle$, where $LA_n = \{\}$ and $W_n = \{A\}$.

This definition says that a derivation of A is a sequence of stages such that in the first stage, no syntactic structure has yet been built, and in the last stage all the lexical items in the initial lexical array have been used up, as in this example:

(6) Derivation of (4)

$S1 = \langle\{John_2,see_1,John_1\},\{\}\rangle \quad \rightarrow$ Select $John_1$
$S2 = \langle\{John_2,see_1\},\{John_1\}\rangle \quad \rightarrow$ Select $see_1$
$S3 = \langle\{John_2\},\{see_1,John_1\}\rangle \quad \rightarrow$ Merge($see_1$,$John_1$)
$S4 = \langle\{John_2\},\{\{see_1,John_1\}\}\rangle \quad \rightarrow$ Select $John_2$
$S5 = \langle\{\},\{John_2,\{see_1,John_1\}\}\rangle \quad \rightarrow$ Merge($John_2$,$\{see_1,John_1\}$)
$S6 = \langle\{\},\{\{John_2,\{see_1,John_1\}\}\}\rangle$

Note that $W_1 = \{\}$, $LA_6 = \{\}$, and $W_6$ contains one element, so this sequence of stages is a derivation of $\{John_2,\{see_1,John_1\}\}$.

Written more succinctly, the derivations of (2) and (4) can be given as follows:

(7) $\langle\ \langle\{see_1,John_1\},\{\}\rangle,\langle\{see_1\},\{John_1\}\rangle,\langle\{\},\{see_1,John_1\}\rangle,\langle\{\},\{\{see_1,John_1\}\}\rangle,$
$\langle\{\},\{\{John_1,\{see_1,John_1\}\}\}\rangle\rangle$

(8) $\langle\ \langle\{John_2,see_1,John_1\},\{\}\rangle,\langle\{John_2,see_1\},\{John_1\}\rangle,\langle\{John_2\},\{see_1,John_1\}\rangle,$
$\langle\{John_2\},\{\{see_1,John_1\}\}\rangle,\langle\{\},\{John_2,\{see_1,John_1\}\}\rangle,\langle\{\},\{\{John_2,\{see_1,John_1\}\}\}\rangle\rangle$

Focusing on (8), notice that other derivations produce the same structure, {John$_2$, {see$_1$,John$_1$}}. For example, we could select *see*$_1$ before selecting *John*$_1$, or we could select all three lexical items before doing any merges. In addition to those possibilities, notice that we could "renumber" the lexical items in infinitely many different ways to obtain essentially equivalent derivations, deriving {John$_1$,{see$_2$, John$_3$}}, {John$_4$,{see$_5$,John$_6$}}, and so on.

## 4. Occurrences

The definition of Merge has the effect that a particular syntactic object can occur two times (or more) in a structure, at different positions. This happens when Merge(A,B) applies and B is contained in A, the subcase of internal Merge. It is often useful to talk about the different occurrences (which occupy different positions) of a particular syntactic object. Related notions will also play a role in our discussion of chains in section 5.

Chomsky (2000a:115) proposes two ways to define the position of an occurrence in a structure. First, he takes "an occurrence of $\alpha$ in K to be the full context of $\alpha$ in K." Alternatively, he suggests a simplification where "an occurrence of $\alpha$ is a sister of $\alpha$." Chomsky does not develop the first definition based on "full context." The second definition, which is based on sisterhood, runs into the problem that the sister of $\alpha$ might also have several occurrences. Therefore, defining the notion of occurrence in terms of sisterhood does not specify (in the general case) a unique position in the structure being built. For example, consider a VP [$_{\text{VP}}$ V DP]. Both the V and the DP can undergo internal Merge. Hence defining the position of V as the sister of DP does not specify a particular position, given that the syntactic object DP has more than one position.

Consider a syntactic object SO = {S$_1$,{S$_1$,S$_2$}}. We say that S$_1$ occurs twice in SO. The position of an occurrence is given by a "path" from SO to the particular occurrence. A path is a sequence of syntactic objects $\langle SO_1,SO_2,\ldots,SO_n\rangle$ where for every adjacent pair $SO_i$,$SO_{i+1}$ of objects in the path, $SO_{i+1} \in SO_i$ (i.e., $SO_{i+1}$ is immediately contained in $SO_i$). With this definition, the two occurrences of S$_1$ in SO are identified by the two different paths that begin with SO and end with S$_1$ (cf. the notion of "dominance path" in Wilder 2008:239 and the notion of "position" in Kracht 1999:261):

(9) Position of highest occurrence of S$_1$ in SO = $\langle$ {S$_1$,{S$_1$,S$_2$}},S$_1$ $\rangle$
(a sequence of two syntactic objects)
Position of lowest occurrence of S$_1$ in SO = $\langle$ {S$_1$,{S$_1$,S$_2$}},{S$_1$,S$_2$},S$_1$ $\rangle$
(a sequence of three syntactic objects)

We define the notions of position and occurrence in a structure as follows:

**Definition 16**

The *position* of $SO_n$ in $SO_1$ is a *path*, a sequence of syntactic objects $\langle SO_1,SO_2,\ldots, SO_n\rangle$ where for all $0 < i < n$, $SO_{i+1} \in SO_i$.

**Definition 17**

B *occurs* in A at position P iff $P = \langle A,\ldots,B\rangle$. We also say B has an occurrence in A at position P (written $B_P$).

Sometimes we will say "an occurrence of X" when we mean "an occurrence of X in position P of syntactic object SO," when the position P and object SO are implicit in the discussion. When talking about a syntactic object A contained in a workspace W, we will define A's position in W with respect to one of the roots (undominated syntactic objects) of W (e.g., A occurs at position P of $B \in W$).

Given these definitions of *position* and *occur*, it is important to revisit the definitions of *sister*, *immediately contain*, *contain*, and *c-command*. These terms are commonly used in the syntax literature for relations between occurrences of syntactic objects. We have already given the definitions of *immediately contain* and *contain* as relations between syntactic objects in Definitions 8 and 9. Reformulating these as relations between occurrences, we have:

**Definition 18**

Let A, B and C be syntactic objects, then, in C, occurrence $B_P$ *immediately contains* occurrence $A_{P'}$ (for paths P, P′ in C) iff $P = \langle X_1,\ldots,X_n\rangle$ and $P' = \langle X_1,\ldots, X_n,X_{n+1}\rangle$.

Note that if B occurs in position $P = \langle X_1,\ldots,X_n\rangle$ in C, and A occurs in position $P' = \langle X_1,\ldots,X_n,X_{n+1}\rangle$ in C, by the definition of paths, it follows that $X_1 = C$, $X_n = B$, $X_{n+1} = A$, and $A \in B$. Clearly, we can relate the *immediately contains* relation between occurrences to the corresponding relation between syntactic objects as follows:

**Theorem 1**

If occurrence $B_P$ immediately contains occurrence $A_{P'}$ in C (for some paths P, P′ in C) then, in C, B immediately contains A. If B immediately contains A, then every occurrence of B immediately contains some occurrence of A.

Similarly for sisterhood, one can define it as a relation between syntactic objects:

**Definition 19**

Let A, B, C be syntactic objects (where $A \neq B$), then A and B are *sisters* in C iff A, $B \in C$.

But a definition corresponding to actual use in the syntax literature makes reference to occurrences:

**Definition 20**

Let A, B, C be syntactic objects (where A ≠ B), then in C, $A_P$ is a *sister* of $B_{P'}$ iff P = $\langle X_1,\ldots,X_{n-1},X_n\rangle$ (where $X_{n-1}$ = C and $X_n$ = A) and P′ = $\langle X_1,\ldots,X_{n-1},X'_n\rangle$ (where $X'_n$ = B).

**Theorem 2**

If in C, $A_P$ is a sister of $B_{P'}$, then A and B are sisters in C.

Similarly, c-command can be defined as a relation between syntactic objects:[7]

**Definition 21**

Let A and B be syntactic objects, then A *c-commands* B in D, iff there is a syntactic object C, such that:

(i) C is a sister of A in D, and
(ii) either B = C or C contains B.

A *asymmetrically c-commands* B iff A c-commands B and A and B are not sisters.

In SO = $\{S_1,\{S_1,S_2\}\}$, according to this definition, $S_1$ c-commands $S_1$. We would usually say that one occurrence of $S_1$ c-commands the other. That is, the occurrence of $S_1$ in position $P_1$ c-commands the occurrence of $S_1$ in position $P_2$ (where positions are defined by paths). The occurrence-based definition is:

**Definition 22**

In D, $A_P$ *c-commands* $B_{P'}$ iff there is an occurrence $C_{P''}$ such that:

(i) $C_{P''}$ is a sister of $A_P$ in D, and
(ii) either $B_{P'} = C_{P''}$ or $C_{P''}$ contains $B_{P'}$, in D.

**Theorem 3**

If in C, $A_P$ c-commands $B_{P'}$, then A c-commands B in C.

## 5. Digression: Syntactic Objects Built from Chains

The indices of lexical tokens allow us to distinguish between the results of internal and external merge, as in (2) and (4), repeated here:

(2) by internal merge: $\{\{John_1,see_1\},John_1\}$
(4) by external merge: $\{\{John_1,see_1\},John_2\}$

[7] Adger (2003:117) offers a similar definition (also cf. Chomsky 2000a:116). Our definition differs from those considered in Barker & Pullum 1990:4, 10, in which command relations are reflexive and it is not the case that "commanders neither dominate commanded nodes nor are dominated by them."

We follow Epstein & Seely (2006:chap. 2) in dispensing with the notion of Chain, giving rise to a multidominance approach based on lexical item tokens. An alternative is to distinguish the result of internal and external Merge by the use of Chains (see Chomsky 1995:253 and also Chomsky 2008). In this section, we briefly compare these two ways of looking at movement.

One could say that in (2) a Chain links the two occurrences of *John* and in (4) there are no nontrivial Chains. If Chain links are explicitly indicated, the indices of lexical item tokens (which distinguish otherwise identical lexical items in a tree) can be dispensed with.

A standard way to define a chain is as a sequence ($n$-tuple) of occurrences (see Chomsky 1995:43 and also Chomsky 2001:39). However, in the course of a derivation, the occurrences (defined by paths) would change each time Merge takes place. Therefore, we have chosen an equivalent formulation that redefines syntactic objects to include the chains formed at each stage in the derivation.

For example, each set {A,B} formed by internal Merge could be indexed with a path from A to B, a "link" to the preceding element B:

(2′) by internal merge: $\{\{\text{John,see}\},\text{John}\}_{\langle\{\text{John,see}\},\text{John}\rangle}$
(4′) by external merge: {{John,see},John}

Various versions of this idea have been proposed in the literature (Nunes 2004:165, n. 15; Kracht 2001; Stabler 2001), but here we briefly develop an especially simple approach for comparison with our token-based perspective.

Because this alternative does not use lexical tokens, we can dispense with lexical arrays, workspaces, and the Select operation, so UG is slightly simplified. Our syntactic objects are pairs SO = $\langle S,P\rangle$, where P is a path, where paths are defined in essentially the same way as before, except that the SOs are pairs:

**Definition 16′**

A sequence of syntactic objects $\langle\langle S_1,P_1\rangle,\ldots,\langle S_n,P_n\rangle\rangle$ is a *path* iff for every adjacent pair $\langle\langle S_i,P_i\rangle,\langle S_{i+1},P_{i+1}\rangle\rangle$, $S_{i+1} \in S_i$. When $n = 0$ we have the trivial path ε. Otherwise, when $n \geq 1$, the path is also called a *position* of $\langle S_n,P_n\rangle$ in $\langle S_1,P_1\rangle$.

**Definition 7′**

A pair $\langle S,P\rangle$ is a *(Chain-based) syntactic object* iff

(i) S is a lexical item and P is a path, or
(ii) S is a set of syntactic objects and P is a path.

We often write $S_P$ for $\langle S,P\rangle$. And when the path P = ε, we often omit it, writing just S.

The definition of Merge has the same form as before but operates on the new syntactic objects:

**Definition 13′**

Given distinct Chain-based syntactic objects A, B, Merge(A,B) = {A,B}.

**Definition 14′**

A *(Chain-based) derivation* from lexicon L is a finite sequence of syntactic objects $\langle SO_1,\ldots,SO_n\rangle$, for $n \geq 1$, where each $SO_i = \langle S_i,P_i\rangle$, such that either

(i) $S_i \in L$, a lexical item, and $P_i = \varepsilon$, or
(ii) $S_i$ = Merge(A,B) = {A,B} where
  (a) A and B appear earlier in the derivation,
  (b) either $P_i$ is a position of B in A or $P_i = \varepsilon$, and
  (c) if $P_i = \langle SO_1,\ldots,SO_k\rangle$, for $k \geq 1$, then $B \notin S_i$ where $SO_i = <S_i,P_i>$ for any $i < k$.

Condition (b) is a disjunction. The first disjunct is called *(Chain-based) internal Merge* and the second disjunct is *(Chain-based) external Merge.* Internal and external Merge both involve exactly the same operation. They differ only in where the argument B is found. Note that (c) is a locality condition. It disallows a path to B that skips a c-commanded B. In the token-based derivation, the question of which occurrence of B is internally merged, when there is more than one, never arises because no indication is provided one way or the other.

**Definition 15′**

A Chain-based syntactic object A is *derivable from lexicon L* iff there is a derivation $\langle SO_1,\ldots,SO_n\rangle$ with $SO_n$ = A.

With these definitions, corresponding to (7) and (8), which are derivations of (2) and (4), respectively, we have these corresponding Chain-based derivations (dropping the empty path, subscript ε, from the "trivial" chains):

(7′) $\langle$John,see,{John,see},{{John,see},John}$_{\langle\{John,see\},John\rangle}\rangle$
(8′) $\langle$John,see,{John,see},{John,{John,see}}$\rangle$

Derivation (8′) is a sequence of four syntactic objects, all trivial chains indexed with ε. Derivation (7′), on the other hand, derives {{John,see},John}$_{\langle\{John,see\},John\rangle}$, which has the nontrivial path $\langle$\{John,see\},John$\rangle$, a position of the internally merged John in the structure {John,see}.

It might seem obvious that these Chain-based derivations are equivalent to our token-based derivations, but that impression is not quite right. Recall that there is a nonidentity requirement on Merge (see Definition 13). Merge(A,B) is only defined if A and B are distinct (but see fn. 5 on self-Merge). The nonidentity requirement on Merge (see Definition 13) has different effects in the token-based and Chain-based accounts. Token-based Merge allows any two distinct elements to Merge, even if one is a renumbering of the other. For example, if A is a lexical item, token-based Merge can derive $\{A_1,A_2\}$. In a system that does not introduce indices on lexical items, there

is no such distinction among lexical items. For example, the nonidentity requirement on Chain-based Merge blocks merging $A_\varepsilon$ with itself to form $\{A_\varepsilon,A_\varepsilon\}_\varepsilon = \{A_\varepsilon\}_\varepsilon$.

The lack of indices on lexical items is the main appeal of the Chain-based system, but it means that, in a sense, the Chain-based merge is more restrictive: there is no way to Merge a lexical item with itself. Which perspective is right for human language?

If the Chain-based restriction is the right one, then the token-based restriction is too weak. On this view, we should strengthen the nonidentity condition of Definition 13 to say that Merge(A,B) is defined only when A and B are distinct under all renumberings.

If the token-based restriction is the right one, then for lexical items $A_1$ and $A_2$ Merge($A_1$,$A_2$) is defined even if $A_1$ is just a renumbering of $A_2$. On this view, the Chain-based nonidentity restriction would be blocking derivations that should be allowed. We could modify Chain-based Merge by removing the distinctness condition. This would allow us to form the nonbranching $\{A_\varepsilon,A_\varepsilon\}_\varepsilon = \{A_\varepsilon\}$, corresponding to the token-based structures $\{A_i,A_j\}$ where $i \neq j$. This might seem reasonable, but notice that the Chain-based system will still be unable to make distinctions that are present in the token-based grammar. For example, in the structure $\{A_i,A_j\}$, $A_i$ and $A_j$ could both potentially undergo internal Merge, each with its own landing site. In sum, even without the distinctness requirement, positions in the Chain-based structures cannot be brought into a unique correspondence with positions in token-based structures. Structure could be added to the Chain-based account to preserve the distinct identities of these parts (cf. Gärtner 2002:sect. 3.3.3), but that would make the Chain-based approach more complex and less appealing.

Given that it is not clear what to do here, let us maintain both definitions of Merge with their distinctness requirements. The correspondence between token-based and Chain-based derivations is then restricted. Merge of token-based elements that are renumberings of each other will have no Chain-based equivalents. But apart from that, the following natural correspondence holds (proof in Appendix):

**Theorem 4**

Let T(Lex) be the set of syntactic objects with derivations in which Merge never applies to two objects A,B where A is a renumbering of B. And let [T(Lex)] be the set of equivalence classes of those objects in T(Lex) with respect to renumbering. Let C (Lex) be the set of syntactic objects with Chain-based derivations. There is a one-to-one, onto, structure-preserving correspondence between [T(Lex)] and C(Lex).

To recap, the key reason that the correspondence must be stated on equivalence classes of structures is that Chain-based structures do not care about the renumberings of token-based structures. If we put renumberings into token-based equivalence classes, then we get a natural one-to-one correspondence between the approaches, up to the different effects of the distinctness condition of the Merge operation.

This theorem represents the natural result that in some sense Chain-based theories and multidominance-based theories are equivalent in the structures that they produce. We return to the token-based formulation in the following sections.

## 6. General Properties of Derivations

This section establishes some basic properties of derivations and then considers four constraints on grammar that have been discussed in the literature. These results hold for all lexicons, so we use notion of being derivable (from some lexicon),

**Definition 23**

A syntactic object is *derivable* iff it is derivable from some Lexicon L. Similarly, a workspace W is *derivable* iff for some Lexicon L, there is some derivation

$\langle\langle LA_1,W_1\rangle,\ldots,\langle LA_n,W_n\rangle\rangle$ from L such that $W = W_n$.

The fact that Merge blocks the formation of unary branching has already been mentioned, but we can be more specific:

**Definition 24**

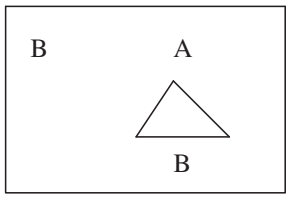
Syntactic object A is *binary branching* iff both A and everything contained in A is either a lexical item token or a syntactic object immediately containing exactly two syntactic objects.

**Theorem 5 (Binary branching)**

Every derivable syntactic object is binary branching.

There are many binary-branching structures that cannot be built, though, and many workspaces that cannot be derived. For example, it is not possible to derive a workspace W = {A,B} where B occurs as a root of W and B also occurs somewhere in A. In other words, B has two occurrences, but they are not in the same syntactic object, the same "tree." Workspaces like (10) are not derivable.

(10) An underivable workspace



There is no way to generate the two occurrences of B in (10). It is easy to state and prove the following general claims (proofs in Appendix):

**Theorem 6**

In every derivable workspace W, if A is a root in W ($A \in W$), then there is no other root $B \in W$ such that A contains an occurrence of B.

**Theorem 7**

A derivable workspace contains two occurrences of A iff either A or some B containing A has undergone internal Merge.

We now turn to four widely discussed conditions that fall out as theorems from our framework: the No-Tampering Condition (NTC), the Extension Condition, Inclusiveness, and Local Economy. These conditions do not filter out unacceptable derivations (which would be the normal interpretation of a constraint or condition in syntactic theory); rather, they make explicit certain properties of the derivations already defined. Syntactic operations and derivations could in principle have been defined in such a way that one or more of these conditions would fail. We will also show how the NTC and the Extension Condition are independent conditions (and so should not be conflated).

Consider first the NTC, which Chomsky (2007:8) defines as follows: "Suppose X and Y are merged. Evidently, efficient computation will leave X and Y unchanged (the No-Tampering Condition [NTC]). We therefore assume that NTC holds unless empirical evidence requires a departure from SMT [strong minimalist thesis] in this regard, hence increasing the complexity of UG. Accordingly, we can take Merge(X, Y) = {X,Y}." As made clear in Chomsky (2005:13), there is a close connection between the NTC and the Copy Theory of Movement: "The no-tampering condition also entails the so-called copy theory of movement, which leaves unmodified the objects to which it applies, forming an extended object."

**Theorem 8**

For any two consecutive stages in a derivation $S_1 = \langle LA_1, W_1 \rangle$ and $S_2 = \langle LA_2, W_2 \rangle$, for all A contained in $W_1$, A is contained in $W_2$.

This says that every syntactic object contained in $W_1$ must find a place in $W_2$. No element of $W_1$ can be destroyed or tampered with.[8] For example, it is easy to see that the trace theory of movement violates the NTC. Suppose A contains B, and A is a root in $W_1$ (a workspace) and Merge(A,B) = {A′,B}, where A′ is exactly the same as A except that the occurrence of B contained in A is replaced by a trace *t*. Then $A \in W_1$

[8] Suppose an operation were introduced that makes of a copy of A, call it A′, to form {A′,{B,A}}. Regardless of whether A′ is identical to A, this does not violate the NTC as we have formulated it here, but one might think that this violates the NTC that was originally intended. Depending on how the copy differs from the original, it could violate Inclusiveness, discussed later, or perhaps another plausible NTC-like constraint could be formulated to block implausible copying operations.

but $A \notin W_2$, nor is there a $C \in W_2$, such that $A \in C$. In fact, A is not contained in $W_2$ at all (only A′ with the trace is).[9]

Consider next the Extension Condition, which demands that the syntactic structures in a workspace be extended by Merge (preventing so-called countercyclic operations). As Chomsky (1995:190) notes: "A second consequence of the extension condition is that given a structure of the form $[_{X'}$ X YP], we cannot insert ZP into X′ (yielding, e.g., $[_{X'}$ X YP ZP]), where ZP is drawn from within YP (raising) or inserted from outside by GT [Generalized Transformation]."

**Theorem 9**

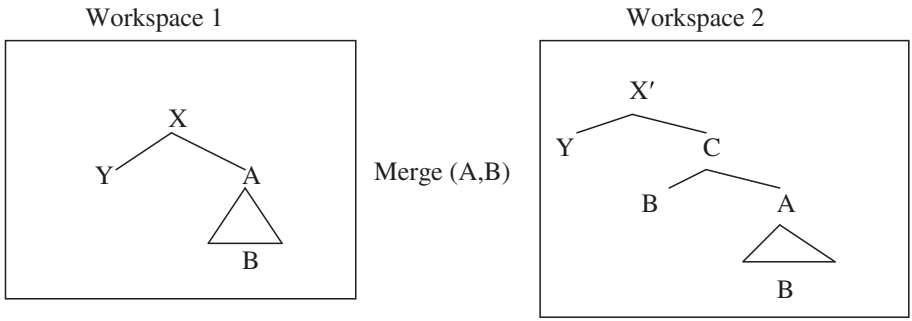
For any two consecutive stages $S_1 = \langle LA_1, W_1 \rangle$ and $S_2 = \langle LA_2, W_2 \rangle$, if $S_1$ derives $S_2$ by Merge, then there is some $A \in W_1$ and $C \in W_2$ such that

(i) $C \notin W_1$ (C is created by Merge)
(ii) $A \notin W_2$ (A is extended)
(iii) $A \in C \in W_2$ (A is extended to form C)

This says that A in $W_1$ is extended to C in $W_2$. On this formulation, the trace theory also violates the Extension Condition. Suppose once again that A contains B, and A is a root in $W_1$ (a workspace) and $C = \text{Merge}(A,B) = \{A', B\}$, where A′ is exactly the same as A except that the occurrence of B contained in A is replaced by a trace *t*. Then in $W_2$, $A \notin C$. (Instead, $A' \in C$.) Therefore, A has not been extended.

In many cases, the NTC and the Extension Condition prohibit the same kinds of illicit derivations. For example, both conditions would prevent defining Merge so as to allow so-called countercyclic movement, as shown in (11). In the derivation, Merge applies countercyclically, forming Merge(A,B) = C (see Collins 1997:84, where it is argued that Merge cannot replace terms).

(11) Derivational diagram of Merge violating NTC and Extension Condition

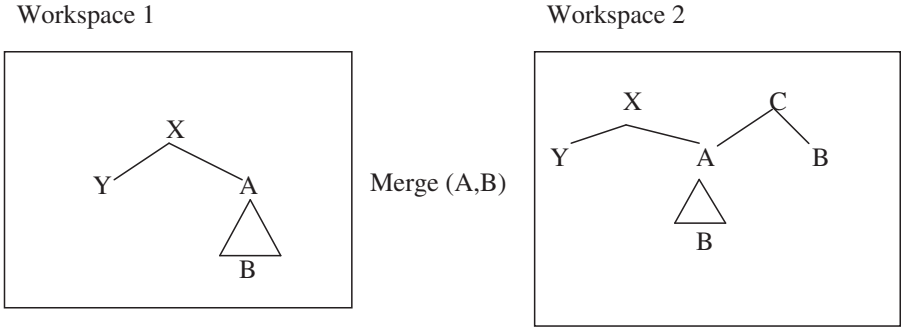



[9] One immediate consequence of the NTC is that the tucking-in derivations of N. Richards (2001:38–46) are not possible. Similarly, Lasnik's (1999:207) claim that A-movement does not leave a trace is inconsistent with the NTC ("A-movement, unlike A′-movement, does not leave a trace, where a trace is, following Chomsky, a copy of the item that moves"). It remains to be seen how the system could be changed to allow these alternatives.

This is not a possible derivation given our definitions, intuitively because it does not "act at the root," as made explicit by the Extension Condition. Furthermore, this operation of Merge tampers with the internal structure of X, violating the NTC.

Other derivations would violate the Extension Condition but not the NTC, which shows that these conditions are conceptually distinct and should not be confused. Consider a slight modification to the countercyclic derivation, where B merges with A, forming C, but C does not replace A.

(12) Derivational diagram of Merge violating Extension Condition but not NTC




Again, this is not a possible derivation, as we have defined it (because $A \notin W_1$; see Definition 14). Note that it violates the Extension Condition: no constituent in $W_1$ is extended (in the sense of a root X becoming a dominated X). However, the derivation does not violate the NTC.

Next, we take up the inclusiveness condition, defined by Chomsky in several works as follows:

> "Another natural condition is that outputs consist of nothing beyond properties of items of the lexicon (lexical features)—in other words, that the interface levels consist of nothing more than arrangements of lexical features." (Chomsky 1995:225)

> Inclusiveness "permits rearrangement of LIs and of elements constructed in the course of derivation, and deletion of features of LI—but optimally, nothing more." (Chomsky 2000a:113)

> Inclusiveness "bars introduction of new elements (features) in the course of computation: indices, traces, syntactic categories or bar levels, and so on." (Chomsky 2001:2–3)

We define inclusiveness in the following way:

**Theorem 10**

In any derivation $(\langle LA_1,W_1\rangle,\ldots,\langle LA_n,W_n\rangle)$ where $W_n = \{A\}$, the only elements contained in $W_n$ are the lexical item tokens from $LA_1$ and syntactic objects containing them.

A discrepancy between Chomsky's version of Inclusiveness and ours is that our version allows indices on lexical item tokens, whereas Chomsky's version does not, a problem noted by Chomsky (1995:227): "*l* and *l'* are marked as distinct for CHL if they are formed by distinct applications of Select accessing the same lexical item of N. Note that this is a departure from the inclusiveness condition, but one that seems indispensable: it is rooted in the nature of language, and perhaps reducible to bare output conditions."

As we observed in the discussion of (2) and (4), the structures in (13) are importantly different, if $i$ is distinct from $m$.

(13) S2 = $\{John_{i,}\{John_i,see_k\}\}$ and
S4 = $\{John_{m,}\{John_i,see_k\}\}$

Here, S2 has two paths to one token of *John*, whereas S4 has two paths to two different tokens of *John*. As discussed, this distinction can be indicated with the indices on the lexical items as in S2 and S4 (as in the representations of the graph structures in (3) and (5)).[10] This distinction is arguably essential at the CI interface (to allow for the representation of the output of internal Merge) and so Inclusiveness must be formulated so as to allow it (which is exactly what we have done in Theorem 10).[11] If the indices on lexical item tokens were eliminated, then some other device would have to distinguish between S2 and S4 in (13). One possibility is to introduce Chains indicated by path indices (see sect. 5); another is to let Merge build more complex graphs (see Gärtner 2002). For a phase-based approach, see Groat 2013.

The last general condition we consider is Local Economy, first proposed by Collins (1997:4) and reformulated slightly here to make it consistent with our terminology:

**Theorem 11**

Given a stage in a derivation $S_i = \langle LA_i, W_i \rangle$, which is part of a derivation $D = \langle S_1, \ldots, S_i, \ldots, S_n \rangle$, whether an operation OP applies to elements of $W_i$ (as part of a derivation) is determined completely by $W_i$ and the syntactic objects it contains.

Suppose that A and B are roots of some workspace $W_1$. According to Local Economy, whether or not Merge applies, forming {A,B}, could not depend on information contained in another workspace (from a stage either earlier or later in the derivation, or from a different derivation altogether). The way we have defined derivations, this result follows trivially. But the point is that we could have defined

[10] See Gärtner 2002 and Kracht 2008 for other approaches to multidominance in directed acyclic graphs like those in (3) and (5).

[11] Kitahara (2000) argues that such distinctness markings are not needed, and hence Chomsky's original formulation of the Inclusiveness Condition, which refers to lexical items (and not lexical item tokens), can be maintained. We simply note that Kitahara's proposed solution only distinguishes distinct pronoun tokens with Case features and was not extended to distinguishing distinct tokens of lexical items in general.

Merge and derivations otherwise, in such a way that Local Economy would not hold.[12]

## 7. Labels

In this section we define a labeling algorithm. We start by defining triggered Merge. Then, we define labels in terms of triggered Merge. We believe we have captured the standard account of labeling of the principles and parameters framework and early minimalism (putting aside adjuncts, which we have not formalized). Future discussions of labeling algorithms could take our formalization as the baseline for comparison.

Some selected quotes from the literature are given here to show some basic ideas about how Merge might be triggered:[13]

> "For an LI to be able to enter into a computation, merging with some SO, it must have some property permitting this operation. A property of an LI is called a *feature*, so an LI has a feature that permits it to be merged. Call this the *edge feature* (EF) of the LI." (Chomsky 2008:139)

> "[T]here is a Last Resort condition that requires all syntactic operations to be driven by (structure-building or probe) features." (Müller 2010:38)

> "I propose that the same Agree relation underlies all instances of Merge." (Boeckx 2008:92)

> "Summarizing, the (syntactic) head of a constituent built up by Merge is the lexical item that projects its features to the new constituent. That lexical item will be the one that triggered application of Merge in the first place by being specified with c-selectional features that need to be checked. All c-selectional features must be checked by applications of Merge." (Adger 2003:96)

We call the features involved in triggering Merge "trigger features." We assume that such features are to be identified with subcategorization features, EPP features, and OP features for movement to Spec,CP (see discussion after Definition 1).

**Definition 25**

A lexical item token $\langle\langle \text{Sem,Syn,Phon}\rangle, i\rangle$ *contains a trigger feature* TF iff some TF $\in$ Syn is a trigger feature.

The following familiar feature sets could be modeled as trigger features:[14]

[12] Local Economy restricts dramatically the information accessible to determine whether an operation applies. Therefore, a better name for the condition might be *Local Determination*. However, we retain the name for historical reasons (it is the original name from Collins 1997).

[13] See also Stabler 1997, Hornstein 1999:78, Collins 2002, and Frampton & Gutmann 2002.

[14] A reviewer comments, "It is not necessarily obvious that sets rather than multisets will be sufficient for Syn. The exact format of 'subcategorization features,' in particular of verbs like German *lehren* ('to teach') requiring two accusative objects, is relevant in this regard. Equally the possibility of multiple EPP features and a treatment of multiple-*wh*-movement would have to be considered here." We leave open this possibility here.

(14) a. T: Syn = {T,[__vP],EPP[15] }
(T requires a specifier and takes a vP complement)
b. Comp: Syn = {Comp,[__TP],OP}
(Comp requires an operator as specifier and takes an TP complement)

We do not assume that the trigger features of (14a,b) are ordered (contra Müller 2010:38, Stabler 1997). This raises the problem of determining what is generated as complement and specifier. For example, what would prevent the following: $[_{TP}$ vP $[_{T'}$ T DP]], where vP is in the specifier of TP, and DP is in the complement (see definitions of complement and specifier in Definitions 32 and 33, respectively). We assume such unattested merging orders are blocked by other constraints (e.g., in the case at hand, internal Merge of DP from vP to the complement of TP would violate the NTC). We leave more careful consideration of this issue for future work.

We provide definitions of Triggers and Triggered Merge as follows. We assume that there is a function Triggers that, for any SO, yields the total set of unchecked TF tokens contained in that SO. Furthermore, because each syntactic object determines its derivational history (and the NTC guarantees that nothing in that history is ever tampered with), we can tell, in every derived structure, which trigger features have been checked. When a syntactic object SO has no trigger features left, Triggers(SO) will be empty.

**Definition 26**

Triggers is any function from each syntactic object A to a subset of the trigger features of A, meeting the following conditions:

(i) If A is a lexical item token with *n* trigger features, then Triggers(A) returns all of those *n* trigger features. (So when $n = 0$, Triggers(A) = {}.)
(ii) If A is a set, then A = {B,C} where Triggers(B) is nonempty, and Triggers(C) = {}, and Triggers(A) = (Triggers(B)) – {TF}, for some trigger feature TF $\in$ Triggers(B).
(iii) Otherwise, Triggers(A) is undefined.

Definition 26 is not the definition of a particular function but of a class of functions, each of which satisfies the three properties listed in Definition 26. Devising a particular Trigger function would depend on such issues as how the EPP and subcategorization features are checked. For example, can a head have two identical trigger features (e.g., two EPP features, two [__DP] features)? How do trigger features impose category constraints and what kinds of category constraints do they impose? How many trigger features can a particular lexical head have? In answering these questions we could devise a particular Trigger function. Therefore, when Triggers(B) has more than one element, we assume that Triggers({B,C}) will check a

[15] On the EPP as a requirement that a clause must have a specifier, see Lasnik 2001:360. For extensive arguments against postulating an EPP feature, see Epstein & Seely 2006.

particular one, determined by B and C, and we leave aside the question of which one it is. However, none of these issues bears on the adequacy of the notion of a Trigger function for our formalism.

We define Triggered Merge as follows, replacing Definition 13:

**Definition 27**

Given any syntactic objects A, B, where Triggers(A) $\neq$ {} and Triggers(B) = {}, Merge(A,B) = {A,B}.

Notice that the triggering condition entails that A,B are distinct. Another consequence is that only one trigger feature can be checked by each Merge operation. As before, this definition of triggered Merge makes no distinction between the two subcases of Merge: internal Merge and external Merge. Lastly, no features are actually deleted, and hence there is no violation of the NTC.

Suppose a lexical item token $see_1$ has two trigger features and token $John_2$ has no trigger features. Then we could have a derivation like this:

(15) Derivation involving lexical item token with two trigger features

a. $\langle\{John_1,see_2\},\{\}\rangle$ → Select $John_1$
b. $\langle\{see_2\},\{John_1\}\rangle$ → Select $see_2$
c. $\langle\{\},\{John_1,see_2\}\rangle$ → Merge($see_2$,$John_1$)
d. $\langle\{\},\{\{John_1,see_2\}\}\rangle$ → Merge($\{John_1,see_2\}$,$John_1$)
e. $\langle\{\},\{\{John_1,\{John_1,see_2\}\}\}\rangle$

This derives the previously discussed structure (2). After the first Merge operation, Triggers($\{John_1,see_2\}$) will have just one feature available. After the second Merge operation, $\{John_1,\{John_1,see_2\}\}$ will not have any trigger features available. That is, Triggers($\{John_1,\{John_1,see_2\}\}$) = {}. The two trigger features of $see_2$ are both unavailable because they were checked by the two Merge operations.

The definition of triggered Merge entails the following asymmetry:

**Theorem 12**

If triggered Merge(A,B) is defined, triggered Merge(B,A) is undefined.

In our approach, the structural relation important for feature checking is sisterhood (created by Merge). There is no reference to either m-command or specifiers in these definitions. In fact, m-command plays no role in our formalization, and specifiers are defined purely in terms of triggered Merge (see below).

Given that Merge is triggered, it is trivial to define syntactic category labels. We will formalize the intuition that the label is always the head that triggers Merge. Some quotes from the literature give background on this approach:

> "Set-Merge of (α,β) has some of the properties of Agree: a feature F of one of the merged elements (say, α) must be satisfied for the operation to take place. . .the label of the selector projects." (Chomsky 2000a:134)

> "Headedness: The item that projects is the item that selects." (Adger 2003:92)

Within this general approach, the question remains as to how to represent the label. We adopt a functional approach. There is a function that has the set of syntactic objects as its domain, and the set of lexical items tokens as its range (see Chomsky 1995:244, 398 on some earlier approaches to labels in the minimalist framework, see Collins 1997:64, who proposed the functional approach to labels, see Collins 2002 for an approach dispensing with labels, and see Seely 2006 on criticisms of earlier minimalist approaches to labels).

**Definition 28**

Label is a syntactic function from syntactic objects to lexical items tokens, defined in the following way:

(i) For all lexical item tokens LI, Label(LI) = LI.
(ii) Let W be a derivable workspace. If {A,B} is contained in W, and Triggers(A) is nonempty, then Label({A,B}) = Label(A).

Label(SO) is often called the *head* of SO. It is now easy to establish (proof in Appendix):

**Theorem 13**

Let A, B, C be syntactic objects. If C = Merge(A,B), then Label(C) = Label(A).

Another general property of labels is that if a constituent B undergoes internal Merge, targeting A, then A projects not B (Chomsky 1995:256). This result follows immediately from our definitions.

Our formalization allows natural definitions of all the X′ Theory concepts:

**Definition 29**

For all C a syntactic object and LI a lexical item token, both contained in a derivable workspace W, C is a maximal projection of LI (written $\text{Max}_W(\text{LI})$) iff Label(C) = LI and there is no D contained in W which immediately contains C such that Label(D) = Label(C).

For example, when $\text{Merge}(\text{see}_1,\text{John}_2) = \{\text{see}_1,\text{John}_2\}$ (which is an element of W), $\text{Label}(\{\text{see}_1,\text{John}_2\}) = \text{see}_1$, and $\text{Max}_W(\text{see}_1) = \{\text{see}_1,\text{John}_2\}$, the maximal projection of $see_1$.

**Definition 30**

For all C, C is a *minimal projection* iff C is a lexical item token.

**Definition 31**

For all syntactic objects C contained in workspace W, LI a lexical item token, C is an *intermediate projection* of LI iff Label(C) = LI, and C is neither a minimal projection nor a maximal projection in W.

The complement is the first element merged with a head, and a specifier is any subsequent element merged with a projection of the head.

**Definition 32**

Y is the *complement* of X in C iff C = Merge(X,Y) and X is a lexical item token.

**Definition 33**

Y is the *specifier* of X in C iff C = Merge(X,Y) where X is not a lexical item token. When LI = Label(X), we also say Y is a *specifier* of LI in C. If XP is the maximal projection of LI, and Y is a specifier of LI, then we also say Y is a specifier of XP, and write Y = Spec XP.

Definition 27 of Triggered Merge and Definition 33 of specifier allow a lexical item to have multiple specifiers. Such multiple specifiers have been empirically argued for in the domain of multiple *wh*-questions (see Richards 2001, among many others).

On this account, there is a close relation between triggered Merge and labels: both are ways to indicate that Merge is asymmetric, and furthermore, the Label function is defined purely in terms of how features are checked. Given this close connection, it may be that one or the other is redundant, which was, in essence, the argument of Collins 2002. We hope the formalization in this section will be useful in ongoing debates about labeling algorithms (see Collins 2002; Seely 2006; Chomsky 2008, 2013; Citko 2008; Cecchetto & Donati 2010; Donati & Cecchetto 2011).

## 8. Transfer

The syntactic objects generated by Merge must be mapped to the interfaces: the conceptual-intentional (CI) interface and the sensorimotor (SM) interface. The operation that does this mapping is called *Transfer* (see Chomsky 2004:107). We will treat Transfer as relativized to a phase P and composed of two operations: $\text{Transfer}_{\text{PF}}$ and $\text{Transfer}_{\text{LF}}$,

**Definition 34**

For syntactic object SO with Triggers(SO) = {}, Transfer(P,SO) = $\langle \text{Transfer}_{\text{PF}}(\text{P,SO}), \text{Transfer}_{\text{LF}}(\text{P,SO}) \rangle$.

$\text{Transfer}_{\text{LF}}$ is the first operation of the semantic component, which maps the SO to a form that can be interpreted by the CI interface. $\text{Transfer}_{\text{PF}}$ (also referred to as

Spell-Out) is the first operation of the phonological component, which maps the SO to a form that can be interpreted by the SM interface. An important question, which we do not address, is where (truth conditional) semantic rules of interpretation and familiar phonological rules fit into this framework.

An important aspect of minimalist syntax is that information interpreted by the interfaces is computed cyclically.[16] Now suppose that at some point in the derivation the syntactic object SO is formed, and Transfer(P,SO) applies. Once a PF sequence is formed (see below), it can never be broken up again in the derivation. In order to permit internal Merge after Transfer, it must be the case that Transfer(P,SO) may leave an escape hatch: "Applied to a phase P, S-O must be able to spell out P in full, or root clauses would never be spelled out. But we know that S-O cannot be *required* to spell out P in full, or displacement would never be possible." (Chomsky 2004:108) We implement this escape hatch with *Cyclic-Transfer*, the first version is given in Definition 36.

Uriagereka (1999:sect. 10.2) discusses the issue of relating "a structure that has already been spelled out to the still 'active' phrase marker." We dub this the *Assembly Problem* and note that it has received little attention in the minimalist literature. Uriagereka sketches several solutions to this problem. On the conservative approach, "the collapsed Merge structure is no longer phrasal, after Spell-Out; in essence, the phrase marker that has undergone Spell-Out is like a giant lexical compound" (p. 256). Our formulation in which $\langle \text{Transfer}_{\text{PF}}(\text{P,Y}), \text{Transfer}_{\text{LF}}(\text{P,Y}) \rangle$ is inserted back into the tree is similar to his conservative approach to this issue. We call this the plug-back-in model. This model has the virtue that the phase impenetrability condition (PIC) becomes a theorem (see Theorem 14). Unfortunately, as our formalization clearly reveals, the plug-back-in model has a number of undesirable properties. First, it requires a complication of the definition of syntactic object (see Definition 37). Second, it entails that derivations involving Cyclic-Transfer violate the NTC. Third, it complicates the treatment of remnant movement (see sect. 11). For these reasons, we believe that the plug-back-in model should be rejected. In section 12, we sketch a non-NTC-violating alternative.

To define *Cyclic-Transfer*, we first need to define *strong phase*. We assume the extensional definition of strong phase heads given in Definition 35. Something like this should eventually follow from more basic assumptions (see Chomsky 2007:19 on the status of TP as a nonphase). Given our formalization of Transfer, it should be possible to compare various definitions of strong phases.[17] We leave the notions "transitive" and "unergative" undefined.

[16] See Uriagereka 1999, which first introduced the notion of Multiple Spell-Out; see Epstein & Seely 2006 for a different conception of cyclic spell-out (one incompatible with Chomsky 2004:122, in particular the discussion before (20)). See Müller 2010:40 for an analysis where every XP is a phase (contra Definition 35). Also see Obata 2010 for discussion. There is need of a critical overview of these various approaches, which we are unable to give for reasons of space.

[17] See Legate 2003:506, where it is argued that "unaccusatives and passive VPs are phases as well"; see Collins 2005:98 for discussion of the phasal status of passives; and see Dobashi 2003 on the relationship between derivation by phase and phonological phrases.

**Definition 35**

A syntactic object SO is a strong phase iff SO is a maximal projection with label LI and either: (a) the syntactic category of LI is Comp, or (b) the syntactic category of LI is transitive or unergative v.

**Definition 36**

For any derivable workspace W where SO is a strong phase and SO ∈ W and A is the complement of the label of the phase, let Cyclic-Transfer(SO) = SO′ where SO′ is obtained from SO by replacing A by $\langle \text{Transfer}_{PF}(SO,A), \text{Transfer}_{LF}(SO,A) \rangle$.

The result of Cyclic-Transfer must feed further syntactic rules. For example, if a *wh*-word moves to Spec,CP, the resulting CP can be embedded under another verb. Given that Merge is only defined for syntactic objects, the result of Cyclic-Transfer must be a syntactic object:

**Definition 37 (replacing Definition 7).**

X is a *syntactic object* iff

(i) X is a lexical item token, or
(ii) X = Cyclic-Transfer(SO) for some syntactic object SO, or
(iii) X is a set of syntactic objects.

With this definition, for example, if SO = {H,XP} and Cyclic-Transfer applies to produce SO′ = {H,Transfer(SO,XP)}, then Transfer(SO,XP) is not a syntactic object but SO′ is.

Lastly, we need to fit the operation Transfer into the derivation, just as we did with Merge and Select.

**Definition 38**

A *derivation* from lexicon L is a finite sequence of stages $\langle S_1,\ldots,S_n \rangle$, for $n \geq 1$, where each $S_i = \langle LA_i, W_i \rangle$, such that

(i) For all LI and $k$ such that $\langle LI,k \rangle \in LA_1$, LI ∈ L,
(ii) $W_1 = \{\}$ (the empty set),
(iii) for all $i$, such that $1 \leq i < n$, one of the following three conditions holds:
(derives by select) for some $A \in LA_i$, $\langle LA_{i+1}, W_{i+1} \rangle = \text{Select}(A, \langle LA_i, W_i \rangle)$, or
(derives by merge) $LA_i = LA_{i+1}$ and the following conditions hold for some A, B:
(a) $A \in W_i$
(b) Either A contains B or $W_i$ immediately contains B, and
(c) $W_{i+1} = (W_i - \{A,B\}) \cup \{\text{merge}(A,B)\}$, or

(derives by transfer) SO ∈ $W_i$ is a strong phase containing no other strong phase whose complement has not yet been transferred, and either

(a) $W_{i+1} = (W_i - \{SO\}) \cup \{Transfer(SO,SO)\}$ (ends the derivation), or
(b) $W_{i+1} = (W_i - \{SO\}) \cup \{Cyclic\text{-}Transfer(SO)\}$.

The basic idea of this approach is to use the workspace as the place where the outputs of Transfer and Cyclic-Transfer are stored, by plugging them back into the syntactic tree.

An important consequence of this definition is that it forces Transfer of a strong phase before it is embedded in another strong phase, because derives-by-Transfer only takes place if SO does not contain any strong phase with an untransferred complement.

With this definition of Transfer, even without the definitions of $Transfer_{PF}$ and $Transfer_{LF}$, we have the PIC as a theorem. The PIC is given by Chomsky (2000a:108) as follows: "In phase α with head H, the domain of H is not accessible to operations outside α, only H and its edge are accessible to such operations." For us, UG (Definition 1) has only three operations. Select applies to a lexical item in the array, so it cannot apply to any complex in the workspace. And Transfer can be regarded as an interface operation. So PIC is relevant only for Merge, and it is a trivial matter to prove (proof in Appendix):

**Theorem 14 (PIC)**

In phase α with head H, in Cyclic-Transfer(α) = {...{H,Z}...} (the output of the Cyclic-Transfer function applied to α), Merge cannot apply to Z or anything contained in Z.

For a phase SO with domain A, Cyclic-Transfer removes A, and replaces it with $\langle Transfer_{PF}(SO,A), Transfer_{LF}(SO,A)\rangle$, which is not a syntactic object and does not contain syntactic objects. As a consequence, the PIC follows as a theorem.

## 9. $Transfer_{LF}$

The next two sections deal with $Transfer_{LF}$ and $Transfer_{PF}$. These sections are necessarily more sketchy and speculative than the preceding material, because minimalist syntacticians have given relatively little attention to the inner workings of Transfer. With this caveat in mind, we start with $Transfer_{LF}$.

The effect of $Transfer_{LF}$ is to strip away the phonetic features and to create a structure where every feature remaining is interpretable at the CI interface (see Chomsky 2000a:118). If any uninterpretable features remain at the point where the CI interface is reached, the derivation will crash (see sect. 13 for definitions of *converge* and *crash*). We make the simplifying assumption that the trigger features are ignored at Transfer and stripped off just like the phonetic features and all the other syntactic features.

We present a bare-bones definition of $Transfer_{LF}$, glossing over many important issues (e.g., the proper representation of movement structures at the CI interface, reconstruction, copy deletion, condition C, and Quantifier Raising, among others):

**Definition 39**

For any derivable workspace W with syntactic object Phase $\in$ W such that Label (Phase) is a strong phase head, for all syntactic objects SO such that either SO = Phase or SO is contained in Phase, $\text{Transfer}_{\text{LF}}$(Phase,SO) is defined as follows:

(a) If SO is a lexical token $\langle\langle \text{Sem,Syn,Phon}\rangle, k\rangle$, $\text{Transfer}_{\text{LF}}(\text{Phase,SO}) = \langle \text{Sem}, k\rangle$.
(b) If SO = {X,Y},
$\text{Transfer}_{\text{LF}}(\text{Phase,SO}) = \{\text{Transfer}_{\text{LF}}(\text{Phase,X}), \text{Transfer}_{\text{LF}}(\text{Phase,Y})\}$.
(c) $\text{Transfer}_{\text{LF}}(\text{Phase}, \langle \text{PHON,SEM}\rangle) = \text{SEM}$.

Clause (a) specifies how lexical item tokens are transferred. Clause (b) specifies how a constituent of the form {X,Y} is transferred. Clause (c) is needed because of Cyclic-Merge. Because $\langle$PHON,SEM$\rangle$ is inserted into the syntactic object being built, later Transfer operations must be able to apply to it. In this preliminary treatment, none of clauses (a)–(c) crucially refer to Phase, but in a more sophisticated treatment that is sensitive to the relations between internal merge and scope, the Phase would be relevant.

The following simplified example shows how $\text{Transfer}_{\text{LF}}$ works, on the assumption that the indicated SO is the VP complement of vP, a phase:

(16) $[_{\text{VP}}$ see Chris]
$\text{LI}_1 = \langle\langle \text{SEE,Syn}_1\text{,/see/}\rangle, 1\rangle$, $\text{LI}_2 = \langle\langle \text{CHRIS,Syn}_2\text{,/chris/}\rangle, 2\rangle$
SO = Merge($\text{LI}_1$,$\text{LI}_2$) = {$\text{LI}_1$,$\text{LI}_2$}
Phase = vP (not shown)
$\text{Transfer}_{\text{LF}}(\text{vP,LI}_1) = \langle \text{SEE},1\rangle$ (deleting Phon and Syn from $\text{LI}_1$)
$\text{Transfer}_{\text{LF}}(\text{vP,LI}_2) = \langle \text{CHRIS},2\rangle$ (deleting Phon and Syn from $\text{LI}_2$)
$\text{Transfer}_{\text{LF}}(\text{vP,SO}) = \{\text{Transfer}_{\text{LF}}(\text{vP,LI}_1), \text{Transfer}_{\text{LF}}(\text{vP,LI}_2)\} = \{\langle \text{SEE},1\rangle, \langle \text{CHRIS},2\rangle\}$

Note that $\text{Transfer}_{\text{LF}}$ preserves both the lexical indices and the hierarchical set structure of the syntactic objects it applies to. The outputs of $\text{Transfer}_{\text{LF}}$ are sent to the CI interface and form the basis of semantic interpretation.

## 10. $\text{Transfer}_{\text{PF}}$

$\text{Transfer}_{\text{PF}}$ deletes any information from a lexical item that cannot be interpreted at the SM interface, including semantic information and syntactic information. Unlike $\text{Transfer}_{\text{LF}}$, the index of the lexical item token is not retained in the output of $\text{Transfer}_{\text{PF}}$. $\text{Transfer}_{\text{PF}}$ constructs a PF sequence by concatenating lexical phonetic features in order.[18]

We formalize the intuition that for economy reasons a syntactic object should, at least in the normal case, only be spelled out once, no matter how many occurrences it has. In this, we agree with Chomsky (2005:13): "If language is optimized for

[18] See Frampton 2004 for a related approach. See Corcoran, Frank & Maloney 1974 for a formal theory of the binary associative, noncommutative operation of concatenation that we indicate with the symbol ^.

satisfaction of interface conditions, with minimal computation, then only one will be spelled out, sharply reducing phonological computation." We put aside issues such as how to handle the copies formed in predicate clefts in which lower occurrences do seem to be spelled-out (see Kandybowicz 2008, Kobele 2006, Hiraiwa 2005).

We first define the notion of *final*, which will be used throughout the definition of $\text{Transfer}_{\text{PF}}$. A syntactic object A in {A,B} is final in a phase P, if {A,B} is not c-commanded by A in P (cf. Gärtner 2002:150–152):

**Definition 40**

A ∈{A,B} is final in SO iff there is no C contained in (or equal to) SO such that A ∈ C, and C contains {A,B}. Otherwise, A is nonfinal in SO.

We are now ready to define Transfer at the PF interface:

**Definition 41**

For any derivable workspace W with syntactic object Phase∈W such that Label (Phase) is a strong phase head, and for all syntactic objects SO such that either SO = Phase or SO is contained in Phase, $\text{Transfer}_{\text{PF}}$(Phase,SO) is defined as follows:

(a) If SO is a lexical item token $\langle\langle \text{Sem,Syn,Phon}\rangle, k\rangle$, then $\text{Transfer}_{\text{PF}}$(Phase,SO) = Phon;
(b) If SO = {X,Y} and X and Y in SO are final in Phase, $\text{Transfer}_{\text{PF}}$(Phase,SO) = $\text{Transfer}_{\text{PF}}$(Phase,X) ^ $\text{Transfer}_{\text{PF}}$(Phase,Y) if either Y is the complement of X, or X is the specifier of Y;
(c) If SO = {X,Y} and X in SO is final in Phase but Y is not, $\text{Transfer}_{\text{PF}}$(Phase,SO) = $\text{Transfer}_{\text{PF}}$(Phase,X);
(d) If SO = {X,Y} where both X and Y in SO are nonfinal in Phase, then $\text{Transfer}_{\text{PF}}$(Phase,SO) = the empty sequence ε.
(e) $\text{Transfer}_{\text{PF}}$(Phase,⟨PHON,SEM⟩) = PHON.

Clause (a) specifies how lexical item tokens are spelled out. Clause (b) entails the order specifier-head-complement is universal, given that no other orderings are provided for (see Kayne 1994 and Aboh 2004). It should be possible to formalize other linear ordering algorithms based on headedness, but we do not explore them here. Clauses (c,d) specify that the lower nonfinal occurrence of a syntactic object is simply ignored at spell-out.[19] One important consequence is that there is no operation like the Chain Reduction of Nunes 2004:27. Clause (e) is needed because of Cyclic-Merge. Because ⟨PHON,SEM⟩ is inserted into the syntactic object being built, later Transfer operations must be able to apply to it.

Examples of $\text{Transfer}_{\text{PF}}$ are given in (17) and (18). In (17), $\text{LI}_2$ is the complement of $\text{LI}_1$, and so /see/ precedes /chris/ (see Definition 41b).

[19] See Kandybowicz 2008:15 for a list of different approaches to the spell-out of occurrences.

(17) $[_{VP}$ see Chris]
$LI_1 = \langle\langle SEE,Syn_1,/see/\rangle,1\rangle$, $LI_2 = \langle\langle CHRIS,Syn_2,/chris/\rangle,2\rangle$
$SO = Merge(LI_1,LI_2)$
Phase = vP (not shown)
$Transfer_{PF}(vP,LI_1) = /see/$, $Transfer_{PF}(vP,LI_2) = /chris/$
$Transfer_{PF}(vP,SO) = Transfer_{PF}(vP,LI_1)$ ^ $Transfer_{PF}(vP,LI_2) = /see/$ ^ /chris/

In the following example, we show how $Transfer_{PF}$ applies to a structure involving internal Merge. We assume *fall* is unaccusative, so that *John* raises from the complement of *fall* to Spec TP.

(18) $[_{TP}$ John T $[_{VP}$ fell <John>] where SO = $[_{VP}$ fell <John>]
$LI_1 = \langle\langle FALL,Syn_1,/fall/\rangle,1\rangle$, $LI_2 = \langle\langle JOHN,Syn_2,/john/\rangle,2\rangle$ (nonfinal)
$SO = Merge(LI_1,LI_2)$
Phase = CP (not shown)
$Transfer_{PF}(CP,LI_1) = /fall/$, $Transfer_{PF}(CP,LI_2) = /john/$
$Transfer_{PF}(CP,SO) = Transfer_{PF}(LI_1) = /fall/$

## 11. Remnant Movement

Remnant movement poses a challenge to the above formulation of Transfer and to the phase-based theory in general, as shown in (21). The problem is formulating $Transfer_{PF}$ in such a way that the gap in the moved remnant is not spelled-out (on alternatives, see Stabler 1997 and Collins & Sabel 2015). Consider the following classic illustration of remnant movement (using <. . .> to denote nonfinal occurrences):

(19) How likely to win is John?

In the following derivation, we omit I-to-C movement:

(20) a. is how likely John to win → Merge
b. John is how likely <John> to win → Merge
c. Comp John is how likely <John> to win → Merge
d. [how likely <John> to win] John is <how likely <John> to win>

In this derivation, Transfer must apply at step (20d). If Transfer applied at step (20c), nothing would be able to undergo internal Merge out of TP, which is the complement of the strong phase head Comp.

Recall Definition 40 regarding final occurrence. Given this definition, consider again (20d): SO = (20d) and A = John and {A,B} = {John,to win}. Now C = TP, where John ∈ TP and TP contains {John,to win}. So John ∈ {John,to win} counts as nonfinal in both occurrences of {John,to win} (the one dominated by AdjP in Spec CP and the one in situ). Hence, the leftmost occurrence of *John* in (20d) is not spelled

out. On the other hand, the occurrence of *John* with the sister {is,{how,{likely, {John$_1$,{to,win}}}}} is final and will be spelled out.

This solution runs up against a problem imposed by the plug-back-in model of Transfer. Once the complement of Comp is transferred, the syntactic structure of the complement is gone. This is what allowed us to derive Theorem 14 (PIC). But if an occurrence A were nonfinal before Transfer, it may become final after Transfer (because relevant structure has been transferred). This problem does not affect (20d), because there is only one phase (the matrix CP) in the structure. However, consider a longer example:

(21) [$_{CP}$ How likely <John> to win do you [$_{vP}$ think [$_{CP}$ that John is]]]

By the time that the matrix clause phase is transferred, the embedded TP will have been transferred, and all the structure relevant at that level for determining whether a *John* is final or nonfinal will be lost.

We take the difficulty in finding a simple analysis of remnant movement to be a lethal problem for the plug-back-in model of Transfer. The problem is fundamental, as we can see by reviewing the assumptions: (i) A phrase SO1 is not altered when some syntactic object SO2 that it contains is merged to a higher position (by the NTC). (ii) The "remnant" SO1 is not altered when it merges to a higher position. These two assumptions have the consequence that Transfer$_{PF}$ cannot know how to spell out the higher occurrence of the remnant SO1 unless it can see lower structure in which the moved SO1 occurs (to see if there are any nonfinal occurrences). So if we assume that the lower structure is removed or replaced as in the plug-back-in model (as described for (21)), the lower structure becomes unavailable to any applications of Transfer$_{PF}$ at higher phases. Therefore, Transfer$_{PF}$ will be unable to spell-out the occurrences correctly. A plug-back-in model can avoid this result only by somehow sacrificing either (i) or (ii), or else by giving up standard assumptions about cyclic Transfer. In the next section, we sketch an alternative.

## 12. NTC-Respecting Transfer$_{PF}$

Given that respecting the NTC is one of the fundamental, motivating ideas of the minimalist approach to grammar,[20] we should consider whether it is possible to define Cyclic-Transfer in a way that respects the NTC and does not require allowing new sorts of syntactic objects. Although we cannot give a full account here, we outline one possibility because the issue has been highlighted by our formalization.

An alternative is to regard Cyclic-Transfer as an operation that does not affect syntactic objects at all but simply affects what is available in the workspace. Instead of thinking of a workspace as containing a set of syntactic objects, all of which are accessible to Merge, we can think of a workspace as providing access to certain occurrences of syntactic objects. One way to do this is to keep a set of syntactic

[20] Nothing definitive can be said about Agree and feature inheritance (Chomsky 2007, Richards 2007) with respect to the NTC until these operations are formalized. We hope our work will make such formalization possible.

objects that have been transferred, and then block all access to those transferred elements. In each step of the derivation, the accessible parts of workspace W are given by W minus those already transferred. This NTC-respecting approach to Cyclic-Transfer would not change the syntactic objects constructed by Merge, and so we do not need the new Definition 37 of syntactic object any more and could return to the original simple Definition 7. Furthermore, such a new account would open the way for a simpler treatment of remnant movement, avoiding the problems mentioned in the discussion after (21).

One thing that the plug-back-in approach gets right is that each syntactic object in a workspace can contain one or more substructures that have already been transferred. We need to keep that association between objects contained in the workspace and their interface outputs. To do that without deleting and replacing the transferred objects, we regard transfer as the establishment of an association between objects in the workspace and interface outputs, or more precisely, as a pair of functions LF, PF. At the beginning of a derivation, the transfer functions LF and PF both have empty domains—that is, they both begin as the empty set of argument-value pairs. In the course of a derivation, they are gradually extended with new argument-value pairs. The list of already transferred elements is given exactly by the domains of these functions. A tentative first formalization of these proposals is briefly sketched in the Appendix. Of course, because this approach does not delete structure, the PIC must come from some stipulation to the effect that although transferred structure is still present, it is not visible to syntactic operations.

## 13. Convergence

The derivation converges when only the pair ⟨PHON,SEM⟩ remains in the workspace, where PHON is interpretable by the SM interface, and SEM is interpretable by the CI interface: "The last line of each derivation D is a pair ⟨PHON, SEM⟩, where PHON is accessed by SM and SEM by C-I. D *converges* if PHON and SEM each satisfy IC [interface condition]; otherwise it *crashes* at one or the other interface." (Chomsky 2004:106).

Minimally, this requirement entails that all material is transferred and that all transferred material is interpretable at the respective interfaces.

### Definition 42

A derivation D = (<$LA_1$,$W_1$>,. . .,<$LA_n$,$W_n$>) where $W_n$ = {<PHON,SEM>} *converges* at the CI interface iff SEM is interpretable at the CI interface. Otherwise, it *crashes* at the CI interface.

### Definition 43

A derivation D = (<$LA_1$,$W_1$>,. . .,<$LA_n$,$W_n$>) where $W_n$ = {<PHON,SEM>} *converges* at the SM interface iff PHON is interpretable at the SM interface. Otherwise, it *crashes* at the SM interface.

**Definition 44**

A derivation *converges* iff it converges at the CI interface and the SM interface. Otherwise, it *crashes*.

## 14. Conclusion

To explain why human languages have movement relations, minimalism proposes that the basic operation, Merge, has in its domain both independently derived structures (external merge) and also the substructures of derived structures (internal merge). This step raises several puzzles, two of which are considered here. First, the results of internal and external Merge are pronounced and interpreted differently, which suggests some difference in their representation. We explored token-based and chain-based proposals about the difference and showed that, although similar on a range cases, they are not identical. We hope that our presentation encourages others to look for empirical consequences of these two ways of looking at movement. We also hope that our investigation will prompt researchers to look for ways to represent the difference between internal and external Merge that do not use either chains or lexical item tokens (see Groat 2013 and Collins & Stabler 2014).

Second, internal Merge raises puzzles about how to enforce the idea that each constituent is transferred to the interfaces just once and how that could connect with phase impenetrability effects. Various versions of Transfer are suggested in the literature. We formalized a plug-back-in model (inspired by Uriagereka's [1999:256] "conservative" approach). As noted, such an approach has several drawbacks (e.g., in the analysis of remnant movement). As an alternative, we proposed an NTC-compliant version in section 12 (and the Appendix).

These preliminary formalizations focus on Merge, Transfer, occurrences, workspaces, and derivations. One lesson from this exercise is that it is not possible to define Merge in isolation, independently from a network of definitions articulating the notion of a derivation. To define Merge and its recursive application properly, we had to define lexical item, lexical item token, syntactic object, a stage in a derivation, the operation Select, and the derive-by-Merge relation between stages, among other notions.

Our formalization assumes that Transfer is an operation in the derivation. An alternative, adopted by researchers working in the Minimalist Grammar framework (see Stabler 1997) is that Transfer is not an operation in the derivation (ordered among Merge operations) but takes place in parallel to the syntactic operations in a derivation. We hope our work prompts further research into the consequences of this choice.

Clearly, our treatment of Transfer is more speculative than the treatment of Merge and raises a number of problems that are not resolved here. Transfer, as opposed to Merge, is a relatively recent addition to minimalist syntax and hence not as well understood.

We did not discuss many other important notions, such as Agree, unvalued features, head movement, and covert movement. In particular, we did not address the important issue of feature representation and the "resource paradox" explored by Gärtner (2002:110) and Nunes (2004). Given a structure formed by internal Merge

{B,{A,B}}, if some features of the higher occurrence of B are checked/deleted, how are those features checked/deleted in the lower occurrence of B? As with Transfer, it is easy to see that there will be NTC-respecting and NTC-violating approaches. We leave a more careful study to future work.

Our investigation has revealed hidden complexity by formally defining notions that have not previously been given formal definitions (e.g., chain, occurrence, workspace, derivation, Transfer). We hope that this exercise leads minimalist syntacticians to try reformulations and alternatives that avoid this complexity and keep to what is essential in order to capture the unbounded capacity that humans have for producing and understanding syntactic structures: Merge(X,Y) = {X,Y}.

## Appendix.

Available from the authors.

## References

Aboh, E.O. 2004. *The morphosyntax of complement-head sequences*. Oxford: Oxford University Press.
Adger, D. 2003. *Core syntax*. Oxford: Oxford University Press.
Adger, D. 2010. A minimalist theory of feature structure. In *Features*, ed. A. Kibort & G.G. Corbett, 185–218. Oxford: Oxford University Press.
Barker, C. & G. Pullum. 1990. A theory of command relations. *Linguistics and Philosophy* 13:1–34.
Boeckx, C. 2008. *Bare syntax*. Oxford: Oxford University Press.
Cecchetto, C. & C. Donati. 2010. On labeling: Principle C and head movement. *Syntax* 13:241–278.
Chomsky, N. 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.
Chomsky, N. 2000a. Minimalist inquiries. In *Step by step: Essays on minimalist syntax in honor of Howard Lasnik*, ed. R. Martin, D. Michaels & J. Uriagereka, 89–155. Cambridge, MA: MIT Press.
Chomsky, N. 2000b. *New horizons in the study of language and mind*. Cambridge: Cambridge University Press.
Chomsky, N. 2001. Derivation by phase. In *Ken Hale: A life in language*, ed. M. Kenstowicz, 1–52. Cambridge, MA: MIT Press.
Chomsky, N. 2004. Beyond explanatory adequacy. In *Structures and beyond*, ed. A. Belletti, 104–131. Oxford: Oxford University Press. (Originally published as MIT Occasional Papers in Linguistics 20. Cambridge, MA: MITWPL, 2001.)
Chomsky, N. 2005. Three factors in language design. *Linguistic Inquiry* 36:1–22.
Chomsky, N. 2007. Approaching UG from below. In *Interfaces + recursion = language?* ed. U. Sauerland & H.-M. Gärtner, 1–29. Berlin: Mouton de Gruyter.
Chomsky, N. 2008. On phases. In *Foundational issues in linguistic theory*, ed. R. Freidin, C.P. Otero & M.L. Zubizarreta, 133–166. Cambridge, MA: MIT Press.
Chomsky, N. 2013. Problems of projections. *Lingua* 130:33–49.
Citko, B. 2005. On the nature of Merge: External Merge, Internal Merge, and Parallel Merge. *Linguistic Inquiry* 36:475–496.
Citko, B. 2008. Missing labels. *Lingua* 118:907–944.
Collins, C. 1994. Economy of Derivation and the Generalized Proper Binding Condition. *Linguistic Inquiry* 25:45–61.
Collins, C. 1997. *Local economy*. Cambridge, MA: MIT Press.

Collins, C. 2002. Eliminating labels. In *Derivation and explanation in the Minimalist Program*, ed. S.D. Epstein & T.D. Seely, 42–64. Oxford: Blackwell.
Collins, C. 2005. A smuggling approach to the passive in English. *Syntax* 8:81–120.
Collins, C. & P. Postal. 2012. *Imposters*. Cambridge, MA: MIT Press.
Collins, C. & J. Sabel. 2015. A C/I-interface condition on remnant movement. In *Remnant movement (Studies in Generative Grammar 123)*, ed. G. Grewendorf, 93–131. Berlin: De Gruyter.
Collins, C. & E. Stabler. 2014. A formalization of a phase-based approach to copies versus repetitions. Ms., New York University, New York, and University of California, Los Angeles.
Corcoran, J., W. Frank & M. Maloney. 1974. String theory. *Journal of Symbolic Logic* 39:625–637.
Dobashi, Y. 2003. Phonological phrasing and syntactic derivation. Ph.D. dissertation, Cornell University, Ithaca, NY.
Donati, C. & C. Cecchetto. 2011. Relabeling heads: A unified account for relativization structures. *Linguistic Inquiry* 42:519–560.
Epstein, S.D., H. Kitahara & T.D. Seely. 2012. Structure building that can't be. In *Ways of structure building*, ed. M. Uribe-Etxebarria & V. Valmala, 253–270. Cambridge: Cambridge University Press.
Epstein, S.D. & T.D. Seely. 2006. *Derivations in minimalism*. Cambridge: Cambridge University Press.
Frampton, J. 2004. Copies, traces, occurrences, and all that: Evidence from Bulgarian multiple-*wh* phenomena. Ms., Northeastern University, Boston.
Frampton, J. & S. Gutmann. 2002. Crash-proof syntax. In *Derivation and explanation in the Minimalist Program*, ed. S.D. Epstein & T.D. Seely, 90–105. Oxford: Blackwell.
Gärtner, H.-M. 2002. *Generalized transformations and beyond*. Berlin: Akademie-Verlag.
Graf, T. 2013. *Local and transderivational constraints in syntax and semantics*. Ph.D. dissertation, University of California, Los Angeles.
Groat, E. 2013. *Is movement part of narrow syntax?*. Handout: Goethe University Frankfurt, Frankfurt, Germany, May, 22.
Guimarães, M. 2000. In defense of vacuous projections in Bare Phrase Structure. In *University of Maryland Working Papers in Linguistics 9*, ed. M. Guimarães, L. Meroni, C. Rodrigues & I. San Martin, 90–115. College Park: Department of Linguistics, University of Maryland.
Harkema, H. 2001. Parsing minimalist languages. Ph.D. dissertation, University of California, Los Angeles.
Hiraiwa, K. 2005. Dimensions of symmetry in syntax. Ph.D. dissertation, MIT, Cambridge, MA.
Hornstein, N. 1999. Movement and control. *Linguistic Inquiry* 30:69–96.
Hunter, T. 2011. Insertion minimalist grammars: Eliminating redundancies between Merge and Move. In *The mathematics of language*, ed. M. Kanazawa, A. Kornai, M. Kracht & H. Seki, 57–71. New York: Springer.
Kandybowicz, J. 2008. *The grammar of repetition*. Amsterdam: John Benjamins.
Kayne, R. 1994. *The antisymmetry of syntax*. Cambridge, MA: MIT Press.
Kitahara, H. 2000. Two (or more) syntactic categories vs. multiple occurrences of one. *Syntax* 3:151–158.
Kobele, G. 2006. Generating copies: An investigation into structural identity in language and grammar. Ph.D. dissertation, University of California, Los Angeles.
Kracht, M. 1999. Adjunction structures and syntactic domains. In *The mathematics of syntactic structure*, ed. H.-P. Kolb & U. Mönnich, 259–299. Berlin: Mouton de Gruyter.
Kracht, M. 2001. Syntax in chains. *Linguistics and Philosophy* 24:467–529.
Kracht, M. 2008. On the logic of LGB-type structures, part I: Multidominance. In *Logics for linguistic structures*, ed. F. Hamm & S. Kepser, 105–142. New York: Mouton de Gruyter.
Lasnik, H. 1999. Chains of arguments. In *Working minimalism*, ed. S.D. Epstein & N. Hornstein, 189–215. Cambridge, MA: MIT Press.

Lasnik, H. 2001. A note on the EPP. *Linguistic Inquiry* 32:356–362.
Legate, J.A. 2003. Some interface properties of the phase. *Linguistic Inquiry* 34:506–516.
Michaelis, J. 2001. On formal properties of minimalist languages. Ph.D. dissertation, University of Potsdam, Potsdam, Germany.
Müller, G. 2010. On deriving CED effects from the PIC. *Linguistic Inquiry* 41:35–82.
Nunes, J. 2004. *Linearization of chains and sideward movement*. Cambridge, MA: MIT Press.
Obata, M. 2010. Root, successive-cyclic, and feature-splitting Internal Merge: Implications for feature inheritance and transfer. Ph.D. dissertation, University of Michigan, Ann Arbor.
Richards, M.D. 2007. Feature inheritance: An argument from the Phase Impenetrability Condition. *Linguistic Inquiry* 38:563–572.
Richards, N. 2001. *Movement in language: Interactions and architectures*. New York: Oxford University Press.
van Riemsdijk, H. 2006. Grafts follow from Merge. In *Phases of interpretation*, ed. M. Frascarelli, 17–44. Berlin: Mouton de Gruyter.
Salvati, S. 2011. Minimalist grammars in the light of logic. In *Logic and grammar*, ed. S. Pogodalla, M. Quatrini & C. Retoré, 81–117. New York: Springer.
Seely, T.D. 2006. Merge, derivational c-command, and subcategorization in a label-free syntax. In *Minimalist essays*, ed. C. Boeckx, 182–217. Amsterdam: John Benjamins.
Stabler, E. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, ed. C. Retoré, 68–95. New York: Springer.
Stabler, E. 2001. Minimalist grammars and recognition. In *Linguistic form and its computation*, ed. C. Rohrer, A. Rossdeutscher & H. Kamp, 327–352. Stanford, CA: CSLI Publications.
Stabler, E. 2010. Computational perspectives on minimalism. In *Oxford handbook of minimalism*, ed. C. Boeckx, 617–641. Oxford: Oxford University Press.
Uriagereka, J. 1999. Multiple spell-out. In *Working minimalism*, ed. S.D. Epstein & N. Hornstein, 251–282. Cambridge, MA: MIT Press.
Veenstra, M. 1998. Formalizing the Minimalist Program. Ph.D. dissertation, University of Groningen, Groningen, the Netherlands.
Wilder, C. 2008. Shared constituents and linearization. In *Topics in ellipsis*, ed. K. Johnson, 229–258. Cambridge: Cambridge University Press.

*Chris Collins*
*New York University*
*Department of Linguistics*
*10 Washington Place*
*New York, NY 10003*
*USA*

*cc116@nyu.edu*

*Edward Stabler*
*University of California, Los Angeles*
*Department of Linguistics*
*3125 Campbell Hall*
*Los Angeles, CA 90095*
*USA*

*stabler@ucla.edu*