A Formalization of Minimalist Syntax

Chris Collins and Edward Stabler

December 2011

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. We would like this formalization to be useful to minimalist syntacticians in formulating new proposals and evaluating their own proposals, both conceptually and empirically.

We do not attempt to formalize all minimalist analyses that have been proposed in the last two decades. Rather, we focus on widely accepted formulations. Where certain alternatives loom large, we mention them briefly. There are many operations we have not treated for reasons of space, including: head movement, Pair-Merge (adjunction), QR Agree and Feature Inheritance. The framework given in this paper could be extended to incorporate various versions of these operations, which could then be compared rigorously.

Our basic approach bears a resemblance to the Minimalist Grammars devised by Stabler (1997) and the work that it has given rise to.[1] Those grammars were simplified to facilitate computational assessment, but here we make an effort to stay close to mainstream formulations.

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), $\subset$ (is a proper subset of). Given any two sets S and T, the set difference S-T=$\{x|\ x\in S,\ x\notin T\}$. And $S\times T$ is the Cartesian product of S and T, that is, the set of ordered pairs $\{\langle a,b\rangle|\ a\in S,b\in T\}$. As usual, free variables in definitions are understood to be universally quantified. For example, "W is a workspace iff…" means the same thing as "For all W, W is a workspace iff…"

1. Preliminary Definitions

Definition 1. Universal Grammar
Universal Grammar is a 6-tuple: <PHON, SYN, SEM, Select, Merge, Transfer>

PHON, SYN and SEM 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. Definition 1 captures what is invariant in the human language faculty. There is no need to augment the 6-tuple with a specification of the format of lexical items, since the required format of lexical items is already given by the definitions of the operations.

We assume that UG specifies three sets of features: semantic features (SEM), phonetic features (PHON), and syntactic features (SYN). We assume that these three sets do not overlap, although in some cases this assumption may be too restrictive (e.g., for

[1] See for example, Michaelis (2001), Harkema (2001), Stabler (2010), Salvati (2011).

phi-features). SEM may include the features like *eventive* or *(not stative)*; it could include thematic roles like *agent, recipient, experiencer*; and it could also include semantic values like *λyλx.x breaks y*. PHON may contain segments, phonological features like [+ATR], and ordering restrictions. SYN includes syntactic categories like N, V, P, Adj, but also subcategorization features, EPP features, and "unvalued" features [uF] valued by Agree.

For the moment, we do not need to be clearer about what the features are, but will 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 of three sets of features,

LI = <Sem, Syn, Phon>

where Sem ⊂ SEM, Syn ⊂ SYN, and Phon ⊂ PHON.

Note that for some lexical items, Sem, Syn or Phon could be empty.

Definition 3. A *lexicon* is a set of lexical items.

While infinite lexicons are not excluded in principle, since the lexical items are basic (not generated) and human minds are finite, only finite lexicons need be considered. Null lexicons are also not excluded in principle.

Definition 4. An I-language is a pair <LEX, UG> where LEX is a lexicon and UG is Universal Grammar.

We make the implicit assumption that any normal adult can access any feature of the universal sets PHON, SEM, SYN. Even if this assumption turns out to be wrong, it is a natural framework to start from.

In order to allow structures in which a given lexical item occurs twice in a structure, the lexical items in our structures will be indexed with integers (this is basically equivalent to Chomsky 1995:227[2], contra Kitahara 2000). These structures with indexed lexical elements have a natural interpretation as graphs where an indexed lexical item corresponds to a leaf node in a graph. For example, in the sentence "the dog saw the other dog", there are two tokens of the single lexical item "dog". 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, other distinguishing marks would serve just as well, e.g., <dog, !>, <dog, !!>, <dog, !!!>,etc. (as long as there could be an unlimited number of such marks).

As discussed below (see Theorem 6), although lexical items are indexed, there is no need for additional co-indexing of elements related by movement, since the lexical

[2]Chomsky (1995: 227): "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 LF. *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 our formalization, the items are always already distinct in the 'lexical array'; see Definition 6.

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 our formalization.

Definition 5. A *lexical item token* is a pair <LI,k> where LI is a lexical item and k is an integer.[3]

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 will 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.

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' in our formalization. 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 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.[4]

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 domains and ranges of the Merge functions (defined in section 2 below) will be sets of syntactic objects. The actual "constituents" built by Merge will be much more restricted (e.g., binary). See Chomsky (1995: 243) for a definition of syntactic object which incorporates the notion of *label*.

We also need some definitions to refer to the relations between syntactic objects.

Definition 8. Let A and B be syntactic objects, then B *immediately contains* A iff $A \in B$.

---

[3] Since the Sem and Phon features of a lexical item are not accessed until Transfer, an alternative would be to treat Sem and Phon as functions defined on the token index Sem(i) (= a set of semantic features) and Phon(i) (= a set of phonetic features). These functions would only be applied at Transfer, and only the token index would appear in the syntactic derivation.

[4] Nor do we define the notion of subarray (see Chomsky 2000: 106, 2001: 11), since it seems to play no role in Chomsky (2007, 2008).

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.

2. Workspaces, Select and Merge

A derivation of a syntactic object is a series of steps 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:

Definition 10. A *stage (of a derivation)* is a pair S = <LA, W>, where LA is a lexical array and W is a set of syntactic objects. In any such stage S, we will call W the *workspace* of S.

A derivation will be defined below 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 will 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 which 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 in our formalization. At any stage in the derivation there is only one workspace. Formalizing the alternative in our framework would not be difficult.

Definition 11. For any syntactic object X and any stage S=<LA,W> with workspace W, if $X \in W$, X is a *root* in W. When X is a root, we will sometimes say simply that X is undominated in W, or when W is understood, simply that X is undominated.

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

Before discussing the operation Merge, we first formalize an operation of selection of lexical item tokens that applies to stages.

Definition 12. Let S be a stage in a derivation S = <LA, W>.
If $LI \in LA$, then Select(LI, S) = $<LA - \{LI\}, W \cup \{LI\}>$

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 non-empty lexical arrays.[5]

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 (Chomsky 2007: 8 and Stabler 1997):

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. This is the basic structure building operation of syntax. The distinctness clause means that no syntactic object A can be merged with itself. In other words, Merge(A,A) is undefined, though different tokens of the same structure can of course be merged. Ultimately, the distinctness clause accounts for the lack of non-branching projections.

Note that there is no distinction made between External-Merge and Internal-merge. These notions are distinguished not by the operation itself, but by the positions of the objects A, B in the stage where Merge is applied. External-Merge corresponds to the case where A, B $\in$W (a workspace). Internal-Merge corresponds to the case where A$\in$W, and A contains B. In the latter case, A is called the "target of movement".

Our definition also allows Sideward-Merge, where A $\in$ W, C $\in$ W, A and C are distinct, and C contains B. A plausible hypothesis is that Sideward-Merge is excluded as a possible operation (contra Nunes 2004) by economy principles: Sideward-Merge involves three elements: A, B, and C (which contains B), instead of only two for the other subcases. We formalize this economy condition by defining derive-by-Merge below so as to block Sideward-Merge.

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

(1) Merge(see_j, $John_k$) = {see_j, $John_k$}

The result of this operation,{see_j, $John_k$}, has no syntactic category label (e.g., VP). The formalism for labels will be developed below, after the operation of "triggered merge" is defined.

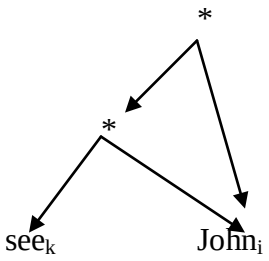
Now suppose that the workspace W is {{$John_i$, see_k}}, then the following Merge operation is also possible:

(2) Merge({$John_i$, see_k}, $John_i$) = {$John_i$, {$John_i$, see_k}}.

[5] Collins (1997: 89-90) and Frampton and Gutman (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 U {LI}. We do not pursue this alternative for reasons of space.

What this means is that when the second argument of Merge, the syntactic object *John*$_i$, occurs inside the first argument, and that object appears in two places in the result. This falls under the Internal-Merge subcase of Merge. Graph theoretically, it is represented as below:

(3) Graph Representation of Merge({John$_i$, see$_k$},John$_i$) = {{John$_i$, see$_k$},John$_i$}



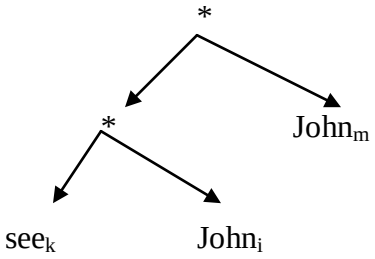
In this kind of "set membership" diagram, the internal nodes labeled * are sets, syntactic objects, with arcs pointing to their elements.

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

(4) Merge({John$_i$, see$_k$},John$_m$) = {{John$_i$, see$_k$},John$_m$ }.

In this structure, there are two distinct lexical item tokens corresponding to the lexical item "John". These lexical item tokens appear in two different positions in the structure. Graph theoretically, it is represented as below:

(5) Graph Representation of Merge({John$_i$, see$_k$},John$_m$) = {{John$_i$, see$_k$},John$_m$ }



A possible alternative approach to the distinction between (3) and (5) is to introduce Chains as fundamental objects (not present in our formalization). Then one could say that in (3) a Chain links the two occurrences of *John* and in (5) there are no non-trivial Chains. Formal approaches along these lines are developed by Kracht (2001) and by Stabler (2001), for example, and we suspect that a structural equivalence could be established between a theory of that sort and the theory developed here. For the rest of this paper we continue to assume that (a) there are lexical item tokens, and (b) there are no Chains.

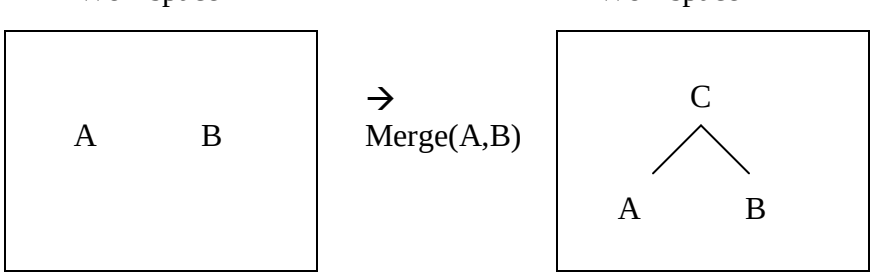
Since we are formalizing derivations in terms of a sequence of stages, we need to show how the operation Merge leads to a transition from one stage to the next[6]:

Definition 14. For any two stages $S_1 = <LA_1, W_1>$ and $S_2 = <LA_1, W_2>$, and any two distinct syntactic objects A, B where $A \in W_1$ and $B \in W_1$ or B is contained in A, S_1 *derives* S_2 *by Merge* iff $W_2 = ((W_1 - \{A, B\}) \cup \{Merge(A,B)\})$. In this case, we will also say that A and B are merged to form {A, B}.

In the above definition, the lexical array does not change from one stage to the next, since it is not altered by Merge (only by Select). If $A, B \in W$ (the External-Merge subcase), then the effect of the derive-by-Merge relation is to remove both A and B from W. If $A \in W$, 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 (since B is not a member of W). Since $B \in W_1$ or B is contained in A, it follows that Sideward Merge is not allowed.

The transition from one workspace to the next can be represented graphically in terms of a *derivational* diagram.

(6) Derivational Diagram of Merge(A,B) ($A \in W, B \in W$)

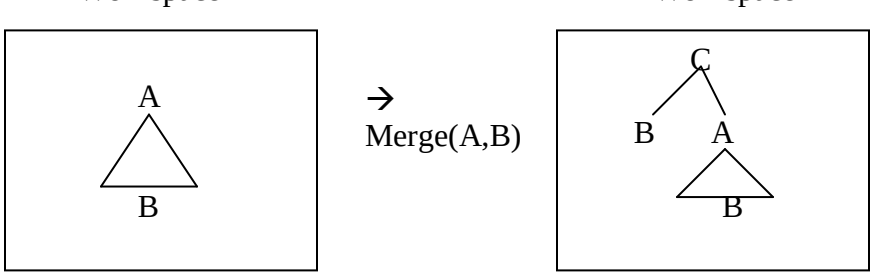


In the derivation in (6), A and B occupy W_1, but are not present in W_2. Furthermore, W_2 has one less element than W_1. Since A does not contain B (so B is external to A), Merge(A,B) is the External-Merge subcase.

In the derivation in (7), by contrast, W_1 and W_2 both have one element. Since B is found internal to A, Merge(A,B) is the Internal-Merge subcase.

[6] For previous work, see Chomsky (1995: 226, 243), Collins (1997: 76), and Frampton and Gutmann (1999). For example, Chomsky (1995: 243) defines the Merge operation as follows: "Suppose a derivation has reached state $\Sigma = \{\alpha, \beta, \delta_1,\ldots,\delta_n\}$. Then application of an operation that forms K as in (5b) [$K = \{\gamma, \{\alpha, \beta\}\}$ – c.c.] converts Σ to $\Sigma' = \{K, \delta_1,\ldots,\delta_n\}$, including K but not α,β." The notion of a sequence of workspaces is not used in Stabler (1997), but is present in Stabler (2006) and developed in Hunter (2011).

(7) Derivational Diagram of Merge(A,B) (A ∈ W, B contained in A)



3. 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.

Chomsky (2000: 115) proposes two ways to define the position of an occurrence in a structure. First, he takes "...an occurrence of α in K to be the full context of α in K." Alternatively, he suggests a simplification where "…an occurrence of α is a sister of α." Chomsky does not develop the first definition based on "full context". The second definition based on sisterhood runs into the problem that the sister of α might also have several occurrences (and hence several sisters). 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 [$_{VP}$ V DP]. Both the V and the DP can undergo movement. Hence defining the position of V as the sister of DP does not specify a particular position, since the syntactic object DP has more than one position. Furthermore, suppose the VP itself undergoes movement (VP….<VP>). Then we have several occurrences of V (one in each occurrence of VP), but the sisterhood definition does not reflect this, since for each V occurrence the sister is the same syntactic object DP.

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 <SO_1,SO_2,...,SO_n> where for every adjacent pair <SO_i, SO_{i+1}> of objects in the path, $SO_{i+1} \in SO_i$ (that is to say, 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:

(8) Position of highest occurrence of S_1 in SO: <{S_1,{S_1,S_2}}, S_1>
(a sequence of two syntactic objects)
Position of lowest occurrence of S_1 in SO: <{S_1,{S_1,S_2}}, {S_1,S_2}, S_1>
(a sequence of three syntactic objects)

We formalize the notions of position and occurrence in a structure with the definitions below.

Definition 15. The *position* of SO_n in SO_1 is a path, a sequence of syntactic objects $<SO_1,SO_2,...,SO_n>$ where for every adjacent pair $<SO_i,SO_{i+1}>$ of objects in the path, $SO_{i+1} \in SO_i$ (SO_{i+1} is immediately contained in SO_i).

Definition 16. B *occurs* in A at position P iff P = <A,...,B>. 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 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, in a way we can now formalize. We have already given the definitions of *immediately contain* and *contain* as relations between syntactic objects in Definitions 8 and 9 above. Restating these relations as relations between occurrences can be done in the following manner:

Definition 17. Let A, B and C be syntactic objects, then, in C, occurrence B_P *immediately contains* occurrence $A_{P'}$ (for any paths P,P' in C) iff P = $<X_1,\ldots,X_n>$ and P'=$<X_1,\ldots,X_n,X_{n+1}>$.

Note that if B occurs in position P=$<X_1,\ldots,X_n>$ in C, and A occurs in position P'=$<X_1,\ldots,X_n,X_{n+1}>$ in C, by the definition of paths, it follows that X_1=C, X_n=B, X_{n+1}=A, and A $\in$ B. So, obviously, 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[7]:

Definition 18. 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 usage in the syntax literature makes reference to occurrences:

[7] It is also possible to formalize the derivational definitions of sisterhood and c-command given in Epstein (1999).

Definition 19. Let A, B, C be syntactic objects (where $A \neq B$), then in C, A_P is a *sister* of $B_{P'}$ iff $P = \langle X_1,\ldots,X_{n-1},X_n\rangle$ (where $X_n = A$) 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 $B_{P'}$ (for some paths P,P' in C) then A and B are sisters in C.

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

Definition 20. Let A and B be syntactic objects, then A *c-commands* B, iff there is a syntactic object C, such that:

i. C is a sister of A, 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. What we would usually say is 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 given below:

Definition 21. 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.

A_P *asymmetrically c-commands* $B_{P'}$ iff A_P c-commands $B_{P'}$ and they are not sisters

Theorem 3. If in C, A_P c-commands $B_{P'}$ (for any paths P,P' in C) then A c-commands B in C.

Given this definition of occurrence, we could define a *Chain* as a sequence of occurrences satisfying some set of conditions (e.g., c-command, locality, the Chain Condition, etc.) of a single syntactic object: $\langle P_1, P_2,\ldots.P_n\rangle$. However, we follow Epstein and Seely (2006, chapter 2) in dispensing with the notion of chain, which will play no role in our formalization.

4. Derivations

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

Definition 22. For any stages, $S_1=\langle LA_1, W_1\rangle$ and $S_2=\langle LA_2, W_2\rangle$, S_1 derives S_2 iff

i. S_1 derives S_2 by Merge,or
ii. for some $LI \in LA_1$, $S_2 = \text{Select}(LI, S_1)$.

Definition 23. A *derivation* from lexicon L is a finite sequence of stages $S_1,\ldots,S_n$, for $n\geq 1$, where each $S_i = <LA_i, W_i>$, such that

i. For all LI and k such that $<LI,k>\in LA_1$, $LI \in L$,
ii. $W_1 = \{\}$ (the empty set),
iii. for all i, such that $1 \leq i \leq n-1$, S_i derives S_{i+1},
iv. $LA_n = \{\}$ (the empty set), and
v. W_n contains exactly one element.

A sequence of stages that satisfies (i-iii) is a *partial derivation*. (So every derivation is a also a partial derivation, but not conversely.) The notion of partial derivation will be useful in proving theorems about derivations and the syntactic objects derived in them.

Definition 23 says that a derivation 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. An example of a derivation is given in (9):

(9) Derivation of "John should like John."

S1 =	$<\{John_1, should_2, like_3, John_4\}, \{\} >$	→ Select $John_4$
S2 =	$<\{John_1, should_2, like_3 \}, \{John_4\} >$	→ Select $like_3$
S3 =	$<\{John_1, should_2 \}, \{like_3, John_4\} >$	→ Merge($like_3$, $John_4$)
S4 =	$<\{John_1, should_2 \}, \{\{like_3, John_4\}\} >$	→ Select $should_2$
S5 =	$<\{John_1\}, \{should_2, \{like_3, John_4\}\} >$	→ Merge($should_2$, {$like_3$, $John_4$})
S6 =	$<\{John_1\}, \{\{should_2, \{like_3, John_4\}\}\} >$	→ Select $John_1$
S7 =	$<\{\}, \{John_1, \{should_2, \{like_3, John_4\}\}\} >$	→ Merge($John_1$, ….)
S8 =	$<\{\}, \{\{John_1, \{should_2, \{like_3, John_4\}\}\}\}>$	

Notice that $W_1 = \{\}$, $LA_8 = \{\}$, and W_8 contains one element, so this sequence of stages is a derivation.

We now establish a few basic properties about what can appear in a partial derivation.

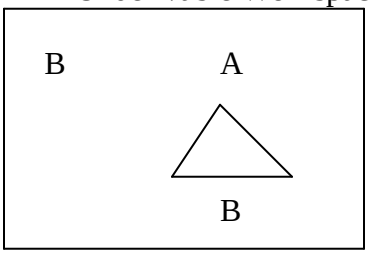
Definition 24. A workspace W is *derivable* iff there is some partial derivation $<<LA_1,W_1>,\ldots,<LA_n,W_n>>$, for $n\geq 1$, such that $W=W_n$. A syntactic object is *derivable* iff it is an element of some derivable workspace.

Definition 25. Syntactic object A is binary branching iff both A and everything contained in A is either a lexical item or a syntactic object immediately containing exactly two syntactic objects.

Theorem 4. (Binary branching) Every derivable syntactic object is binary branching.

The definitions given so far allow the possibility of 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". However, in derivable workspaces, this will never happen. An example of an underivable workspace is shown below:

(10) An Underivable Workspace



Intuitively, given Merge there is no way to generate the two occurrences of A in (10). It is easy to state the general claim and prove it:

Theorem 5. (Uniqueness of root occurrences) In every derivable workspace W, if A is an undominated root in W ($A \in W$), then there is no root $B \in W$ such that A contains an occurrence of B.

Proof. We establish this theorem by proving a stronger claim, namely, that in every partial derivation ($<LA_1,W_1>,\ldots,<LA_n,W_n>$), for every $A \in LA_n \cup W_n$ there is no $B \in LA_n \cup W_n$ such that A contains an occurrence of B. We use an induction on partial derivation lengths. Suppose (n=1). By the definition of "partial derivation", in every partial derivation the first workspace $W_1=\{\}$ so $LA_1 \cup W_1 = LA_1$, and the theorem holds since no lexical item contains any other. Now let the inductive hypothesis (IH) be that the result holds for partial derivations up to length k, for any $k \geq 1$, and we show that this property is preserved in partial derivations ($<LA_1,W_1>,\ldots,<LA_k,W_k>,<LA_{k+1},W_{k+1}>$) of length k+1. We distinguish 3 cases according to how the last step from $<LA_k,W_k>$ to $<LA_{k+1},W_{k+1}>$ is derived.

Suppose first that this last step is derived by Select. In this case, $LA_k \cup W_k = LA_{k+1} \cup W_{k+1}$, and so the theorem holds trivially.

As a second case, suppose $<LA_k,W_k>$ derives $<LA_{k+1},W_{k+1}>$ by the External-Merge subcase of Merge. In this case,

$$LA_{k+1} \cup W_{k+1} = LA_k \cup ((W_k - \{A,B\}) \cup \{Merge(A,B)\}).$$

Since by IH, A and B occur uniquely in $LA_k \cup W_k$, and since they also occur uniquely in $\{A, B\}$ (a subset of W_k which becomes an element of W_{k+1}), it follows that they occur uniquely in $LA_{k+1} \cup W_{k+1}$.

The third possibility is that $<LA_k,W_k>$ derives $<LA_{k+1},W_{k+1}>$ by the Internal-Merge subcase of Merge. In this case also,

$$LA_{k+1} \cup W_{k+1} = LA_k \cup ((W_k - \{A, B\}) \cup \{Merge(A,B)\}).$$

Since by IH, no $C \in LA_k \cup W_k$ has any occurrence in A, and since by assumption B occurs in A, it follows that no $C \in LA_k \cup W_k$ has any occurrence in {A,B}, and hence uniqueness is preserved in $LA_{k+1} \cup W_{k+1}$. QED.

Theorem 6. A derivable workspace contains two distinct occurrences of A iff either A or some B containing A has undergone the Internal-Merge subcase of Merge.

Proof. Again, this is established by induction on the length of partial derivations ($<LA_1,W_1>,\ldots,<LA_n,W_n>$). Suppose (n=1), then since $W_1 = \{\}$ the result holds vacuously. Now let the inductive hypothesis (IH) be that the result holds up to length k, for any $k \geq 1$ and consider any partial derivation ($<LA_1,W_1>, \ldots,<LA_k,W_k>,<LA_{k+1},W_{k+1}>$).

Suppose first that W_k derives W_{k+1} by Select. This operation simply moves a lexical item token LI from LA_k into W_k to produce W_{k+1}. LI cannot occur in W_k already, since any selected element is removed from the lexical array (and all elements of the initial lexical array are distinct). So in W_{k+1} there is just one occurrence of LI, and all other occurrences of elements are unchanged. So the result holds for W_{k+1} in this case.

Now suppose $<LA_k,W_k>$ derives $<LA_{k+1},W_{k+1}>$ by the External-Merge subcase of Merge; again it is clear that the result of this operation will contain multiple occurrences of a constituent iff its arguments do, and so by the IH we know that W_{k+1} will have distinct occurrences iff it has an element A that has undergone Internal-Merge previously.

Finally, consider the case where W_k derives W_{k+1} by the Internal-Merge subcase of Merge. By the definition of Merge(C, A), this step produces D = {A,C} where C contains A. In this case there will be (at least) two different paths to A in D, therefore there will be two occurrences of A in D. This exhausts the possibilities, so the result will hold for W_{k+1}. QED

5. General Theorems about Derivations

In this section we will formulate four very general theorems about derivations: the No Tampering Condition, 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), but rather they make explicit certain properties of the derivations already defined. Syntactic operations such as Merge and the derive-by-Merge relation 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 No Tampering Condition and the Extension Condition are independent conditions (and so should not be conflated).

Consider first the No Tampering Condition, 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 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 No Tampering Condition 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 7. (No Tampering Condition)
For any two consecutive stages in a derivation $S_1 = <LA_1, W_1>$ and $S_2 = <LA_2, W_2>$, for all $A \in W_1$, either $A \in W_2$, or there is some $C \in W_2$ and $A \in C$.

What this says in plain English is that every syntactic object in W_1 must find a place in W_2. No element of W_1 can be destroyed or tampered with. This theorem is easy to prove with an induction of the sort used for the previous results.

To give a simple example, the trace theory of movement violates the No Tampering Condition. 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 $A \in W_1$ but $A \notin W_2$, nor is there a $C \in W_2$, such that $A \in C$. The reason is that A is not contained in W_2 at all (only A' with the trace is)[8].

Consider next the Extension Condition, which demands that structures be extended by Merge. As Chomsky (1995: Chapter 3, 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."

Theorem 8. (Extension Condition)
For any two consecutive stages $S_1 = <LA_1, W_1>$ and $S_2 = <LA_2, W_2>$, if S_1 derives S_2 by Merge, then there is some $A \in W_1$ and $C \in W_2$ such that

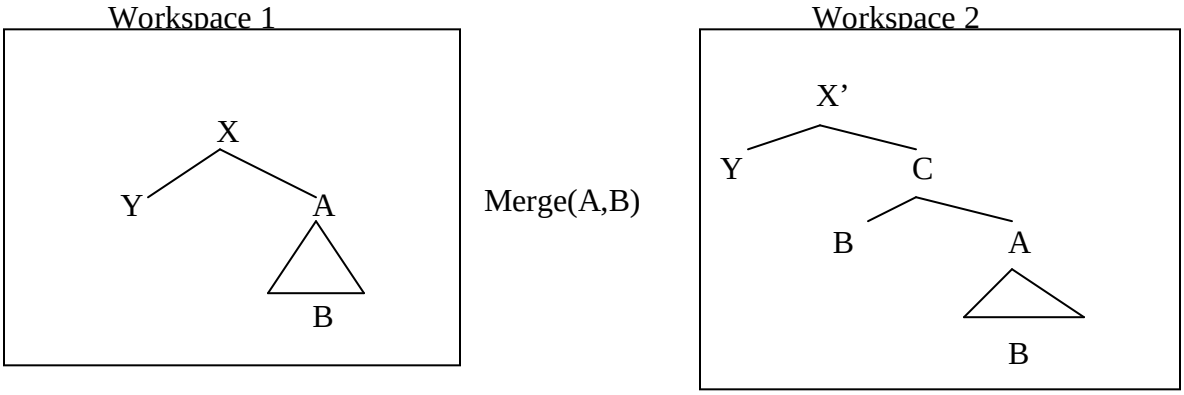
i.	$C \notin W1$	(C is created by Merge)
ii.	$A \notin W2$	(A is extended)
iii.	$A \in C$.	(A is extended to form C)

In plain English this says that A in W_1 is extended to C in W_2.

In many cases, the No Tampering Condition 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 counter-cyclic movement, as illustrated in the derivational diagram below. In the derivation illustrated, Merge applies counter-cyclically, forming Merge(A, B) = C.

[8] One immediate consequence of the definition of Merge (and hence the NTC) is that the tucking-in derivations of 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 definition of Merge and 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 Merge could be redefined to allow these alternatives.

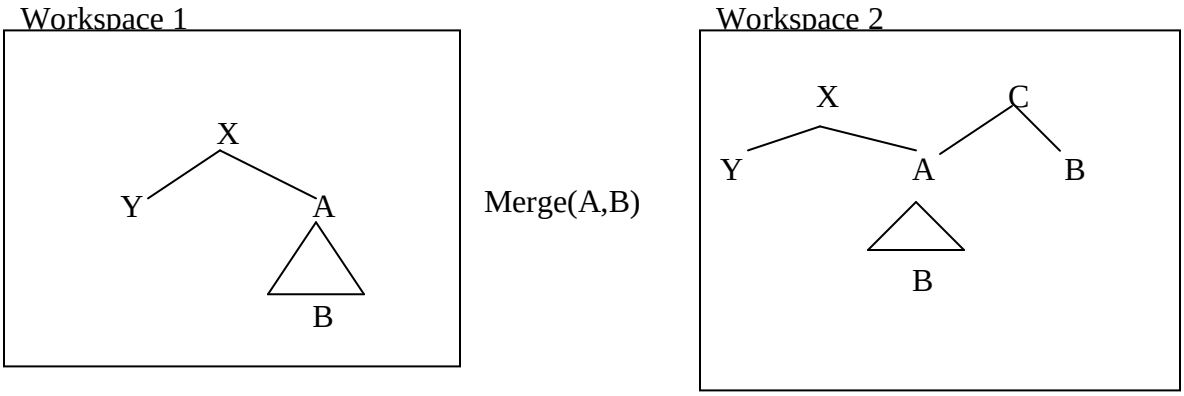
(11) Derivational Diagram of Merge Violating NTC and Extension Condition



This is not a possible derivation (given the definition of the derive-by-Merge relation) since it does not "act at the root", as is made explicit by the Extension Condition. Furthermore, this operation of Merge tampers with the internal structure of X, violating the NTC.

There are other derivations that would violate the Extension Condition, but not the No Tampering Condition, showing that these conditions are conceptually distinct, and should not be confused. Consider a slight modification to the counter-cyclic derivation above, 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 given the definition of the derive-by-Merge relation. Note that it violates the Extension Condition: no constituent in W_1 is extended (in the sense of an undominated X becoming a dominated X). However, the derivation does not violate the No Tampering Condition.

Next, we take up the inclusiveness condition, defined by Chomsky in several places 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 2000: 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 formalize inclusiveness in the following way:

Theorem 9. (Inclusiveness)
In any derivation ($<LA_1,W_1>,...,<LA_n,W_n>$) where $W_n=\{A\}$, the only elements contained in W_n are the lexical item tokens from LA_1 and sets containing them.

A seeming 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 already 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 graphs (3) and (5) above, the structures

(13) $S3 = \{John_i, \{John_i, see_k\}\}$ and
$S5 = \{John_m, \{John_i, see_k\}\}$

are importantly different: S4 has two paths to one token of *John*, while S5 has two paths to two different tokens of *John*. This distinction can be indicated with the indices on the lexical items as in S3 and S5 (as in the representations of the graph structures shown in (3) and (5) above).[9] This distinction is obviously essential at the CI Interface and so Inclusiveness must be formulated so as to allow it (which is exactly what we have done in Theorem 9).[10] If the indices on lexical item tokens were eliminated, then some other device would have to distinguish between S3 and S5 in (13). One possibility is to let

[9] See Gärtner (2001) and Kracht (2008) for other approaches to "multidominance" in directed acyclic graphs like those in (3) and (5), above.

[10] 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 build graphs rather than sets; another possibility mentioned in section 2 is to introduce Chains.

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

Theorem 10. (Local Economy)
Given stage in a derivation $S_i = <LA_i, W_i>$, which is part of a derivation $D = <S_1,\ldots S_i,\ldots S_n>$, whether or not 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.

For example, suppose that A and B are roots of some workspace W_1. Then, 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 derive-by-Merge, this result follows trivially. But the point is that we could have defined Merge and derive-by-Merge otherwise, in such a way that Local Economy would not hold.

6. Labels

In this section we define a labeling algorithm. We start by first 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. Recent discussions of labeling algorithms could take our formalization as the baseline for comparison.

Some selected quotes from the literature are given below illustrating some basic ideas about how Merge might be triggered:[11]

> "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

[11] See also Stabler (1997), Hornstein (1999: 78), Collins (2003) and Frampton and Gutmann (2002).

with c-selectional features that need to be checked. All c-selectional features must be checked by applications of Merge." (Adger 2003: 96).

We will 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.

Definition 26. A lexical item token LI = <<Sem, Syn, Phon>, i> *contains a trigger feature* TF iff some TF∈Syn is a trigger feature.

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

(14) a. Infl: Syn = {Infl, [__vP], EPP[12]}
(Infl requires a specifier and takes a vP complement)
b. Comp: Syn = {Comp, [__IP], OP}
(Comp requires an operator as specifier and takes an IP complement)

We provide simple definitions of Triggers and Triggered Merge as follows. We assume that there is a function Triggers which for any SO, yields the total set of unchecked TF tokens contained in that SO. Furthermore, since 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 27. Triggers is a function from each derivable 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={B,C}, Triggers(B) is nonempty, and Triggers(C)={}, then Triggers(A)=(Triggers(B))-{TF}, for some trigger feature token TF.

When Triggers(B) has more than one element, we assume that Triggers({B,C}) will delete a particular one, determined by B and C, and we leave aside the question of which one it is.

Definition 28. (Triggered Merge, replacing Definition 13)
Given any two distinct syntactic objects A, B where Triggers(A)≠{}and Triggers(B)={}, Merge(A,B)= {A,B}.

[12] 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 and Seely 2006.

Although we state the triggering conditions as part of Merge, it would also be possible to define them as applying at the interfaces to rule out combinations that are not well-formed.

An important consequence of this definition is that only one trigger feature can be checked by each Merge operation. Furthermore, our definition of triggered Merge makes no distinction between the two subcases of Merge: Internal-Merge and External-Merge.

Suppose a lexical item token *see*$_1$ has 2 trigger features, and token *John*$_2$ has 0 trigger features. Then we could have a derivation like this:

(15) Derivation involving LI token with 2 TFs

a. $<\{John_1, see_2\},\{\}>$ $\rightarrow$ Select $John_1$

b. $<\{see_2\},\{John_1\}>$ $\rightarrow$ Select see_2

c. $<\{\},\{John_1, see_2\}>$ $\rightarrow$ Merge(see_2,$John_1$)

d. $<\{\},\{\{John_1, see_2\}\}>$ $\rightarrow$ Merge($\{John_1, see_2\}$, $John_1$,)

e. $<\{\},\{\{John_1,\{John_1, see_2\}\}\}>$

This derives the structure shown in (3) above. After the first merge operation, Triggers($\{John_1, see_2\}$) will have just 1 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 11. If triggered Merge(A,B) is defined, 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 the above definitions. In fact, m-command plays no role in our formalization at all, 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 2000: 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 will adopt a functional approach. There is a function that has the set of derivable syntactic objects as its domain, and the set of lexical items as its range (see Chomsky 1995: 244, 398 on some earlier approaches to labels in the minimalist framework, see Collins 1997: 64 who first proposed the functional approach to labels, see Collins 2002 for an approach dispensing with labels, see Seely 2006 on criticisms of earlier minimalist approaches to labels).

Definition 29. (Label)
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 non-empty, then Label{A, B} = Label(A).

Theorem 12. (Endocentricity)
Let A, B, C be syntactic objects. If C = Merge(A, B), then Label(C) = Label(A).

> Proof. This is immediate from our definitions. Merge(A,B) is defined only if Triggers(A) is non-empty and Triggers(B) is empty. So by the definition of Label, the element that triggers the merge, the one with the non-empty set of available trigger features, is the one that projects. So Label(Merge(A,B))=Label(A) (whenever <A,B> is in the domain of Merge). QED

Endocentricity entails that Merge cannot build structures like these:

(16)

Definition 30. (Maximal Projection)
For all C a syntactic object and LI a lexical item token, both contained in a workspace W, C is a maximal projection of LI (written Max_W(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 Merge(see_1, $John_2$) = {see_1, $John_2$}, Label({see_1, $John_2$}) = see_1, and {see_1, $John_2$}=Max(see_1), the maximal projection of *see_1*.

Definition 31. (Minimal Projection)
For all C, C is a *minimal projection* iff C is a lexical item token.

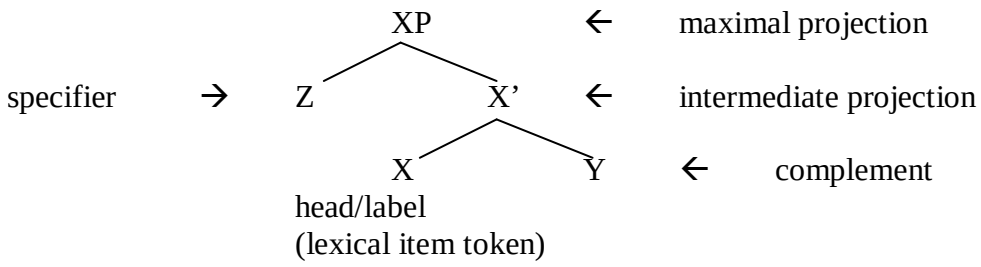
Definition 32. (Intermediate Projection)
For all C, D syntactic objects in workspace W, LI a lexical item token, C is an *intermediate projection* of LI iff Label(C) = LI, and C is neither a minimal projection nor a maximal projection in W.

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

Definition 34. (Specifier)
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 the *specifier* of LI in C.

(17)



On the above 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. In essence, this was the argument of Collins (2002). We hope the formalism given in this section will be of use to ongoing debates about labeling algorithms (see Collins 2002, Seely 2006 and Chomsky 2008).

7. Transfer

The syntactic objects generated by Merge must be mapped to the interfaces: the Conceptual-Intentional Interface and the Sensorimotor Interface. The operation that does this mapping is called *Transfer* (see Chomsky 2004: 107). We will treat Transfer as composed of two operations: Transfer_{PF} and Transfer_{LF}.

Definition 35. (Transfer)
For every syntactic object SO (with Triggers(SO) = {}), Transfer(SO) = <Transfer_{PF}(SO), Transfer_{LF}(SO)>.

Transfer_{LF} is the first operation of the semantic component, which maps the SO to a form that can be interpreted by the CI Interface. Transfer_{PF} 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 will 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.[13] Now suppose that at some point in the derivation the syntactic object SO is formed, and Transfer(SO) applies. After this, nothing further can be extracted from SO. Once a PF sequence is formed (see below), it can never be broken up again in the derivation. In order to permit movement after Transfer, it must be the case

[13] See Uriagereka 1999 who first introduced the notion of Multiple Spell-Out; see Epstein and Seely 2006 for a different conception of cyclic spell-out (one incompatible with Chomsky 2004: 122, in particular the discussion above example (20)). See Müller 2010: 40 for an analysis where every XP is a phase. See Obata (2010) for recent discussion. There is need of a critical overview of these various approaches.

that Transfer(SO) may leave an escape hatch for movement: "Applied to a phase PH, S-O must be able to spell out PH in full, or root clauses would never be spelled out. But we know that S-O cannot be *required* to spell out PH 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. We modify the definition of Cyclic-Transfer in section 10 below to deal with remnant movement.

Definition 36. (Cyclic-Transfer, first version)
For any derivable workspace W = {SO} where SO is a strong phase and A is the complement of the head of the phase, let Cyclic-Transfer$_P$(SO) = SO' where SO' is obtained from SO by replacing A by <Transfer$_{PF}$(A), Transfer$_{LF}$(A)>.

Uriagereka (1999, section 10.2) discusses the issue of relating "…a structure that has already been spelled out to the still 'active' phrase marker." Our formulation where <Transfer$_{PF}$(Y), Transfer$_{LF}$(Y)> is inserted back into the tree is similar to his "conservative" approach to this issue. Unfortunately, it has the property that it violates the NTC. In section 11, we briefly consider a non-NTC violating alternative.

The result of Cyclic-Transfer must feed further syntactic rules. For example, if a wh-word moves to Spec CP (forming a Phase), the resulting CP can be embedded under another verb. Since 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(XP)}, then Transfer(XP) is not a syntactic object but SO' is.

We will assume the extensional definition of strong phase heads given in Definition 38 below. Something like this should eventually follow from more basic assumptions. Given our formalization of Transfer below it should be possible to compare various definitions of strong phases.[14]

Definition 38. (Strong Phase)
A syntactic object SO is a strong phase iff its head X is Comp or X is v* (active transitive or active unergative v).

Lastly, we need to fit the operation Transfer into the derivation, just as we did with for Merge with the derive-by-Merge relation between stages.

[14] See Legate (2003) who argues that "unaccusatives and passive VPs are phases as well"; see Collins (2005:98) for discussion of the phasal status of passives; see Dobashi (2003) on the relationship between derivation by phase and phonological phrases.

Definition 39. (Derives by Transfer)
For any two stages $S_1 = <LA_1, W_1>$ and $S_2 = <LA_1, W_2>$, where $SO \in W_1$ is a strong phase containing no other strong phase whose complement has not yet been transferred, S_1 *derives* S_2 *by Transfer* iff

i. $W_2 = (W_1 - \{SO\}) \cup \{Transfer(SO)\}$ (ends the derivation), or
ii. $W_2 = (W_1 - \{SO\}) \cup \{Cyclic\text{-}Transfer(SO)\}$.

The basic idea of the above approach is to use the workspace as the place where the outputs of Transfer and Cyclic-Transfer are stored, by plugging them directly 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, since derives-by-Transfer only takes place if SO does not contain any strong phase with an untransferred complement.

8. $\textbf{Transfer}_{\textbf{LF}}$

The next two sections deal with $Transfer_{LF}$ and $Transfer_{PF}$. These sections are necessarily more sketchy and speculative than the preceding material, since 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 2000: 118). If any uninterpretable features remain at the point where the CI interface is reached, the derivation will crash (see section 12 for a definition 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.

Definition 40. ($Transfer_{LF}$) For any derivable workspace W with syntactic object $Phase \in W$ such that Label(Phase) is a strong phase head, for all occurrences of objects SO such that either SO=Phase or SO is contained in Phase, $Transfer_{LF}(SO)$ is defined as follows:

a. If SO is a lexical item token <<Sem, Syn, Phon>, k> ,
   $Transfer_{LF}(SO) = <<Sem>, k>$,
b. If $SO = \{X, Y\}$, $Transfer_{LF}(SO) = \{Transfer_{LF}(X), Transfer_{LF}(Y)\}$.[15]
c. $Transfer_{LF}(<PHON, SEM>) = SEM$

Clause (a) specifies how lexical item tokens are interpreted. Clause (b) specifies how a constituent of the form {X, Y} is interpreted. Clause (c) is needed because of

[15] We do not formalize the copy deletion approach to reconstruction. See Chomsky (1995, Chapter 3: 203) "In trace position, the copy of what remains in the operator position deletes,…"; Barss(2002: 693, fn. 3); Fox (2003: 45-47) on the possibility that Trace Conversion is implicit in the semantic rule of interpretation. Furthermore, we do not attempt to incorporate covert QR into our system (see Fox 2003 for an overview of the properties of QR).

Cyclic-Merge. Since <PHON, SEM> is inserted into the syntactic object being built, later Transfer operations must be able to apply to it.

The following simplified example illustrates how $Transfer_{LF}$ works, on the assumption that the indicated SO is the VP complement of vP, a phase:

(18) $[_{VP}$ see Chris]
LI_1 = <<SEE, Syn_1, /see/>, 1>
LI_2 = <<CHRIS, Syn_2, /chris/>,2>
SO = Merge(LI_1,LI_2) = {LI_1,LI_2}
$Transfer_{LF}(LI_1)$ = <SEE, 1> (deleting Phon and Syn from LI_1)
$Transfer_{LF}(LI_2)$ = <CHRIS, 2> (deleting Phon and Syn from LI_2)
$Transfer_{LF}(SO)$ = {$Transfer_{LF}(LI_1)$, $Transfer_{LF}(LI_2)$} = {<SEE, 1>, <CHRIS, 2>}

Note that $Transfer_{LF}$ preserves both the lexical indices and the hierarchical set structure of the syntactic objects it applies to. The output of $Transfer_{LF}(SO)$ will be sent to the CI interface, and form the basis of semantic interpretation.

9. $Transfer_{PF}$

$Transfer_{PF}$ deletes any information from a lexical item that cannot be interpreted at the SM-Interface, including semantic information and syntactic information. Unlike $Transfer_{LF}$, the index of the lexical item token is not needed for $Transfer_{PF}$. $Transfer_{PF}$ constructs a PF sequence by concatenating lexical phonetic features in order.[16]

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 satisfaction of interface conditions, with minimal computation, then only one will be spelled out, sharply reducing phonological computation." Lexical item tokens that are part of a non-final occurrence will simply not be spelled-out by $Transfer_{PF}$ defined below. We put aside issues such as how to handle the copies formed in predicate cleft (see Kandybowicz 2008, Kobele 2007, Hiraiwa 2005).

Definition 41. (Non-Final Occurrences)
The occurrence Y_P in SO is *non-final* in SO iff there is another occurrence $Y_{P'}$ in SO that c-commands Y_P.

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

Definition 42. ($Transfer_{PF}$) For any derivable workspace W with syntactic object Phase$\in$W such that Label(Phase) is a strong phase head, and for all occurrences of objects SO such that either SO=Phase or SO is contained in Phase, $Transfer_{PF}(SO)$ is defined as follows:

[16] See Frampton (2004) for a related approach. See Corcoran et al. (1974) for a formal theory of the binary associative, non-commutative operation of concatenation that we indicate with the symbol ^.

a. If SO is a lexical item token <<Sem, Syn, Phon>, k>, which is a final occurrence in Phase, then $Transfer_{PF}(SO)$ = Phon;
b. If SO={X,Y} and occurrences X and Y are final in Phase, $Transfer_{PF}(SO)$ = $Transfer_{PF}(X)$^$Transfer_{PF}(Y)$ if either Y is the complement of X, or X is the specifier of Y;
c. If SO={X,Y} and occurrence X is final in Phase but occurrence Y is not, $Transfer_{PF}(SO)$ = $Transfer_{PF}(X)$;
d. If SO is non-final in Phase, or SO={X,Y} where both X and Y are non-final in Phase, then $Transfer_{PF}(SO)$ = the empty sequence.
e. $Transfer_{PF}$(<PHON, SEM>) = PHON

Clause (a) specifies how lexical item tokens are spelled out. Clause (b) entails the order Specifier-Head-Complement is universal, since 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 have not explored them here. Clauses (c,d) specify that the lower non-final occurrence of a syntactic object is simply ignored at spell out[17]. 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. Since <PHON, SEM> is inserted into the syntactic object being built, later Transfer operations must be able to apply to it.

An example of $Transfer_{PF}$ is given below:

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

(20) $[_{IP}$ John Infl $[_{VP}$ fell <John>] where SO = $[_{VP}$ fell <John>]
(assume "fall" is unaccusative, so that "John" raises from the complement of "fall" to Spec IP)
LI_1 = <<FALL, Syn_1, /fall/>, 1>
LI_2 = <<JOHN, Syn_2, /john/>, 2> (LI_2 is non-final)
SO = Merge(LI_1,LI_2)
Phase = CP (not shown)
$Transfer_{PF}(LI_1)$ = /fall/
$Transfer_{PF}(LI_2)$ = /john/
$Transfer_{PF}(SO)$ = $Transfer_{PF}(LI_1)$ = /fall/

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

(21) [Who do you think John will see], consider the embedded clause after "who" raises to the embedded Spec CP, but before it raises to the matrix Spec CP:
$[_{CP}$ who John will see]

Phase = {who_1, {$Comp_2$ {$John_3$, {$will_4$, {see_5, who_1 }}}}}
SO = complement of $Comp_2$ = {$John_3$, {$will_4$, {see_5, who_1 }}}
Cyclic-Transfer(Phase) =
{who_1, {$Comp_2$, <$Transfer_{PF}$(SO), $Transfer_{LF}$(SO)>} =
{who_1 {$Comp_2$, </john/^/will/^/see/,
{<JOHN, 3>, {<WILL, 4>, {<SEE, 5>, <WHO, 1>}}}>}}

Given the definition of Transfer, it is now possible to prove the phase impenetrability condition (PIC) as a theorem. The PIC is given by Chomsky (2000: 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." Our UG (Definition 1) has only 3 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:

Theorem 13.(PIC) In phase α with head H that has sister (i.e. complement) occurrence XP_P, in Cyclic-Transfer(α), Merge cannot apply to the sister of H or anything contained in the sister of H.

Proof. By Definition 38, the sister of H in Cyclic-Transfer(α) is Transfer(XP), which is not a syntactic object and does not contain any syntactic objects. Since Merge is only defined for syntactic objects, it cannot apply to the sister of H or to anything contained in the sister of H. QED

Note that in Theorem 13, PIC intuitively is telling us about the accessibility of a particular occurrence XP_P of the syntactic object XP. The definitions given so far allow XP (or something contained in XP) to merge as the specifier of H before Cyclic-Transfer applies, in which case the specifier occurrence would be accessible in Cyclic-Transfer(α), even though neither the complement occurrence nor any syntactic object occurrence inside the complement is present any more.

10. Multiple Transfer at the Phase Level

Remnant movement poses a challenge to the above formulation of Transfer, and to the phase-based theory in general. Although the proposals in this section are preliminary, we take them as a first step in trying to incorporate remnant movement into a phase-based theory and hope that they will stimulate further research.

Consider the following classic illustration (<…> denotes non-final occurrences):

(22) How likely to win is John?

(23) 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 is John <how likely <John> to win>

If this derivation is to converge (as defined in section 12 below), Transfer must apply at step (23d). If Transfer applied at step (23c), nothing would be able to move out of the IP which is the complement of the strong phase head Comp. But if Transfer applies at step (23d), then the simple definition of non-final occurrence in Definition 41 does not yield the intended result. There are three occurrences of *John* in (23d), and under the definition of non-final occurrence the leftmost occurrence counts as final (since there is no other occurrence c-commanding it), and hence should be spelled out. This issue will affect any theory trying to incorporate remnant movement into a phase-based theory.

A natural solution to this problem is to assume that once internal Merge takes place, the lower (non-final occurrence) is spelled out as phonetically zero immediately. Then, when the remnant moves in (23d), there will be no issue of blocking the leftmost occurrence of *John* from spelling out as a final occurrence since it will have already been spelled out as zero. This is the approach of Collins and Sabel 2007 and Stabler 1997, and it is implicit in the "trace" theory of movement rules. However, in the phased-based approach, Transfer takes place at the strong phase head, and not necessarily following every instance of internal Merge.

As a first step in resolving the issue in a phase-based model we assume the following:

(24) At strong phase SO, the operation Cyclic-Transfer can apply more than one time.

We formalize this in the following way:

Definition 36'.(Cyclic-Transfer, second version)
For any derivable workspace W = {SO} where SO is a strong phase and the occurrence A_P (with position P in SO) is the complement of the head of the phase or is contained in the complement of the head of the phase, let Cyclic-$Transfer_P$(SO) = SO' where SO' is obtained from SO by replacing A in position P by <$Transfer_{PF}$(A), $Transfer_{LF}$(A)>.

The way that this modifies the earlier version of Cyclic-Transfer is that it allows the complement of the phase head or any constituent contained in the complement of the phase head to be spelled out, not just the complement of the phase head. The concrete consequence for remnant movement is that Cyclic-Transfer can apply twice (or more) at the level of a single strong phrase head, allowing the interleaving of Merge and Cyclic-Transfer operations at the level of a strong phase. In particular, if there are two occurrences of a constituent (created by Merge) in a phase, the lower one can be spelled out as phonetically zero by Cyclic-Transfer. The derivation of remnant movement is sketched below. In the derivation below, the output of Transfer is abbreviated with hyphenated a-b-c notation, we omit I-to-C movement:

(25) a. Merge({is, {how, {likely, {$John_1$, {to, win}}}}}, $John_1$) =
{$John_1$, {is, {how, {likely, {$John_1$, {to, win}}}}}}

b. Merge(Comp, {John$_1$, {is, {how, {likely, {John$_1$, {to, win}}}}}}) =
{Comp, {John$_1$, {is, {how, {likely, {John$_1$, {to, win}}}}}}}

c. Cyclic-Transfer({Comp, {John$_1$, {is, {how, {likely, {John$_1$, {to, win}}}}}}}) =
{Comp, {John$_1$, {is, {how, {likely, {<∅, JOHN$_1$>, {to, win}}}}}}}

d. Merge({how, {likely, {<∅, JOHN$_1$>, {to, win}}}}, {Comp, {John$_1$, {is, {how, {likely, {<∅, JOHN$_1$>, {to, win}}}}}}}) =
{{how, {likely, {<∅, JOHN$_1$>, {to, win}}}}, {Comp, {John$_1$, {is, {how, {likely, {<∅, JOHN$_1$>, {to win}}}}}}}}

e. Cyclic-Transfer({{how, {likely, {<∅, JOHN$_1$>, {to, win}}}}, {Comp, {John$_1$, {is, {how, {likely, {<∅, JOHN$_1$>, {to, win}}}}}}}}) =
{{how, {likely, {<∅, JOHN$_1$>, {to, win}}}}, {Comp John-is}}

f. Transfer({{how, {likely, {<∅, JOHN$_1$>, {to, win}}}}, {Comp John-is}}) = how-likely-to-win-John-is

This yields the desired result because the Cyclic-Transfer in (25c) applies not to the complement of Comp but to the lower occurrence of *John*, at a point when that occurrence is still c-commanded by the higher one. The remnant wh-movement then applies to the *how likely* phrase that contains the lower occurrence of *John*, now appropriately spelled out as phonetically zero.

This strategy will produce the desired results as long as we can require that lower occurrences of moved elements are transferred before any containing constituent is moved. In other words, what forces the derivation in (25), as opposed to one where remnant movement takes place and the leftmost occurrence of *John* is spelled out in (23d)?

We can think of a number of solutions to this issue. The simplest approach is to define Transfer$_{PF}$ so that it will not apply if there are two final occurrences of a single syntactic object. For example, in (23d), there are two final occurrence of *John*, so Transfer$_{PF}$ will be undefined. To implement this formally, (41d) could be changed to

Definition 42'. (Transfer$_{PF}$)

c. If SO={X,Y} and occurrence X is the unique final occurrence of X in Phase but occurrence Y is not, Transfer$_{PF}$(SO) = Transfer$_{PF}$(X);

Another approach is based on the intuition that the illegitimate application of Transfer$_{PF}$ in (23d) would spell out the phonological features of *John* twice. An economy condition stating that the Phon features of each lexical item can be accessed for Transfer$_{PF}$ exactly once would block the illegitimate derivation. A preliminary attempt to formalize this intuition would modify the definition of Transfer$_{PF}$ in the following way:

Definition 42''. (TransferPF)

a. If SO is a lexical item token <<Sem, Syn, Phon>, k>, which is an unmarked final occurrence in the Phase, then TransferPF(SO) = Phon and the lexical item is marked as spelled out.

We assume that the lexical item itself (i.e. all occurrences) are marked by $Transfer_{PF}$, and (ii) that the derivation will crash when Transfer is required but cannot apply.

11. NTC and Transfer: An Alternative

Since respecting the NTC is one of the fundamental, motivating ideas of the minimalist approach to grammar, 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. One 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 occurrences that have been transferred, and then block all access to those transferred elements. We call the set of transferred occurrences CI/SM (named for the conceptual-intentional and sensorimotor interfaces), so that at each step of the derivation the accessible parts of workspace W are given by W minus the occurrences in CI/SM. When we add an occurrence SO_p to CI/SM, let's write CI/SM $\cup$ $\{SO_p\}$ to represent the addition of occurrence SO_p and all the occurrences contained in SO_p. We will not spell out this perspective in full detail, but Cyclic-Transfer would be defined as follows (mimicking closely the revised Definition 36):

Definition 36''. (NTC-respecting version)
Consider any derivable workspace W={SO} with inaccessible occurrences CI/SM, where SO is a strong phase and the occurrence A_P (with position P in SO) is the complement of the head of the phase or is contained in the complement of the head of the phase.
Then let $Cyclic\text{-}Transfer_P(W,CI/SM) = (W,CI/SM')$ where CI/SM' is obtained from CI/SM by adding

i. A_p and all occurrences contained in A_p, and
ii. for every lexical item token occurring in A_p, the PHON of all occurrences of that lexical item token.

This NTC-respecting approach to Cyclic-Transfer does not change the syntactic objects constructed by Merge, and so we don't need the new Definition 37 of syntactic object any more, and can return to the original simple Definition 7. And Cyclic-Transfer now does not change the syntactic structure at all, but affects only which parts of derived structures are accessible in the workspace. With this second perspective on Transfer, only Merge constructs syntactic objects, and no other operation changes them in any way. Transfer and Select are operations that act on the workspace to affect what is under consideration as the structure is built by Merge.

Interestingly, though the difference between the NTC-violating Definition 36 and the NTC-respecting version of 36'' is important for the overall picture of how the syntax works, it is, remarkably, irrelevant to most of the details of the framework. Either an occurrence is replaced by a pair $<Transfer_{PF}(A_P), Transfer_{LF}(A_P)>$ and some lexical PHON properties are deleted (NTC-violating Cyclic-Transfer), or the occurrence is put

into CI/SM and so is not accessible (NTC-respecting Cyclic-Transfer). With either approach, some parts of syntactic objects are rendered inaccessible, and that is what matters in the details. Theorem 13 (PIC) follows in both cases.

12. Convergence

A derivation converges if in the workspace only the pair <PHON, SEM> remains, 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; otherwise it *crashes* at one or the other interface." (Chomsky 2004: 106).

Minimally, this requirement entails that the result of Transfer_{LF}(SO) should only contain features interpretable at the CI Interface, and that Transfer_{PF}(SO) should only contain features interpretable at the SM Interface.

Definition 43. (Converge at CI Interface)
A derivation D = ($<LA_1, W_1>,\ldots,<LA_n, W_n>$) where W_n = {<PHON, SEM>} *converges* at the CI Interface iff for every feature F contained in SEM, F is interpretable at the CI Interface. Otherwise, it *crashes* at the CI Interface

Definition 44. (Converge at SM Interface)
A derivation D = ($<LA_1, W_1>,\ldots,<LA_n, W_n>$) where W_n = {<PHON, SEM>} *converges* at the SM Interface iff for every feature F contained in PHON, F is interpretable at the SM Interface. Otherwise, it *crashes* at the SM Interface.

Definition 45. (Converge)
A derivation *converges* iff it converges at the CI Interface and the SM Interface. Otherwise, it *crashes*.

13. Conclusion

In this paper we have given a preliminary formalization of minimalist syntax, including the operations Merge and Transfer. 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. In order 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, the derive-by-Merge relation between stages, among other notions.

Clearly, our treatment of Transfer is more speculative than the treatment of Merge, raising a number of problems that have not been resolved here. This corresponds to the fact that Transfer, as opposed to Merge, is a relatively recent addition to minimalist syntax and hence not as well understood. The operations of Transfer_{PF} and Transfer_{LF} need much attention. On the PF side, we have not given a satisfactory treatment of the spell-out of occurrences in predicate cleft and related constructions (see Kandybowicz 2008, Boskovic and Nunes 2007, and Nunes 2004). On the LF side, we have not given a satisfactory treatment of reconstruction effects. We hope that our formalization will

stimulate further efforts in these areas.

One result of our work is to bring up broader theoretical issues that have not received much attention. Should we use lexical item tokens, or Chains, or more complicated graphs? Do we obtain a more revealing and more plausible theory when Transfer is formulated to respect the NTC? We leave these matters for future work.

References:

Aboh, Enoch Oladé. 2004. The Morphosyntax of Complement-Head Sequences. Oxford: Oxford University Press.

Adger, David. 2003. *Core Syntax*. Oxford, Oxford University Press.

Barss, Andrew. 2002. Syntactic Reconstruction Effects. In Mark Baltin and Chris Collins, The Handbook of Contemporary Syntactic Theory. Oxford: Blackwell.

Boeckx, Cedric. 2008. *Bare Syntax.* Oxford: Oxford University Press.

Boskovic, Zeljko and Jairo Nunes. 2007. The Copy Theory of Movement: A View from PF. In Norbert Cover and JairoNunes (eds), *The Copy Theory of Movement*, pgs. 13-74. Amsterdam: John Benjamins Publishing Company.

Chomsky, Noam. 1990. On Formalization and Formal Linguistics. *Natural Language and Linguistic Theory* 8.1, pgs. 143-147.

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

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

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

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

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

Chomsky, Noam. 2007. Approaching UG from Below. In Uli Sauerland and Hans-Martin Gärtner (eds.), *Interfaces + Recursion = Language?* Pgs., 1-29. Mouton de Gruyter, Berlin.

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

Collins, Chris. 1994. Economy of Derivation and the Generalized Proper Binding Condition. *Linguistic Inquiry* 25.1, 45-61.

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

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

Collins, Chris and Joachim Sabel. 2007. An LF-Interface Constraint on Remant Movement. Ms., NYU and Université Catholique de Louvain.

Corcoran, John, William Frank, and Michael Maloney. 1974. String Theory. *Journal of Symbolic Logic* 39.4 , pgs. 625-637.

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

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

Epstein, Samuel David and T. Daniel Seely. 2002. Rule Applications as Cycles in a Level-free Syntax. In Samuel David Epstein and T. Daniel Seely (eds.), *Derivation and Explanation in the Minimalist Program*, pgs. 65-89. Oxford, Blackwell.

Epstein, Samuel David and T. Daniel Seely. 2006. *Derivations in Minimalism*. Cambridge, Cambridge University Press.

Fox, Danny. 2003. On Logical Form. In Randall Hendrick (ed.), *Minimalist Syntax*, pgs. 82-123. Oxford: Blackwell.

Frampton, John. 2004. Copies, Traces, Occurrences and all that: Evidence from Bulgarian Mutiple-Wh Phenomena. Ms., Northeastern University

Frampton, John and Sam Gutmann. 1999. Cyclic Computation, a Computationally Efficient Minimalist Syntax. *Syntax* 2.1, pgs 1-27.

Frampton, John and Sam Gutmann. 2002. Crash-Proof Syntax. In Samuel David Epstein and T. Daniel Seely (eds.), *Derivation and Explanation in the Minimalist Program*, pgs. 90-105. Oxford, Blackwell.

Gärtner, Hans-Martin. 2001. *Generalized Transformations and Beyond.* Akademie-Verlag: Berlin.

Harkema, Henk. 2001. *Parsing Minimalist Languages*. Ph.D. Dissertation, UCLA.

Hiraiwa, Ken. 2005. *Dimensions of Symmetry in Syntax.* Ph.D. Dissertation, MIT

Hornstein, Norbert. 1999. Movement and Control. *Linguistic Inquiry* 30.1, 69-96.

Hunter, Tim. 2011. Insertion Minimalist Grammars: Eliminating Redundancies between Merge and Move. In M. Kanazawa, A.Kornai, M. Kracht and H. Seki, (eds.), *The Mathematics of Language*. New York: Springer.

Kandybowicz, Jason. 2008. *The Grammar of Repetition.* Amsterdam: John Benjamins Publishing Company.

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

Kayne, Richard. 2005. *Movement and Silence*. Oxford: Oxford University Press

Kitahara, Hisatsugu. 2000. Two (or More) Syntactic Categories vs. Multiple Occurrences of One. *Syntax* 3.3, pgs. 151-158.

Kobele, Gregory. 2007. *Generating Copies: An Investigation into Structural Identity in Language and Grammar.* Doctoral Dissertation, UCLA.

Kracht, Marcus. 2001. Syntax in Chains. *Linguistics and Philosophy 24*: 467-529.

Kracht, Marcus. 2008. On the logic of LGB-type structures, Part I: Multidominance. In F. Hamm and S. Kepser (eds.) *Logics for Linguistic Structures*. NY: Mouton de Gruyter.

Lasnik, Howard. 1999. Chains of arguments. In S. Epstein and N. Hornstein (eds.) *Working Minimalism*. MIT Press, pgs. 189-215.

Lasnik, Howard. 2001. A Note on the EPP. *Linguistic Inquiry* 32.2, pgs. 356-362.

Lasnik, Howard. 2003. *Minimalist Investigations in Linguistic Theory*. New York: Routledge.

Legate, Julie Anne. 2003. Some Interface Properties of the Phase. *Linguistic Inquiry* 34.3, pgs. 506-516.
Michaelis, Jens. 2001. *On Formal Properties of Minimalist Languages.* Ph.D. Dissertation, University of Potsdam.
Müller, Gereon. 1998. *Incomplete Category Fronting*. Dordrecht: Kluwer Academic Publishers.
Müller, Gereon. 2010. On Deriving CED Effects from the PIC. Linguistic Inquiry 41.1: 35-82.
Nunes, Jairo. 2004. *Linearization of Chains and Sideward Movement*. MIT Press, Cambridge.
Obata, Miki. 2010. Root, Successive-Cyclic and Feature-Splitting Internal Merge: Implications for Feature Inheritance and Transfer. Doctoral Dissertation, University of Michigan.
Richards, Norvin. 2001. *Movement in Language: Interactions and Architectures.* New York: Oxford University Press.
Richards, Marc D. 2007. Feature Inheritance: An Argument from the Phase Impenetrability Condition. *Linguistic Inquiry* 38.3: 563-572.
Rizzi, Luigi. 1977. The Fine Structure of the Left Periphery. In Haegeman, Liliane (ed.), *Elements of Grammar*, 281-337. Dordrecht: Kluwer Academic Publishers.
Salvati, Sylvain. 2011. Minimalist Grammars in the Light of Logic. In S. Pogodalla, M. Quatrini and C. Retoré (eds.), *Logic and Grammar*.New York, Springer.
Seely, T. Daniel. 2006. Merge, Derivational C-Command, and Subcategorization in a Label-Free Syntax. In Cedric Boeckx (ed.), *Minimalist Essays*. Amsterdam, John Benjamins.
Stabler, Edward. 1997. Derivational Minimalism. In Christian Retoré (ed.), *Logical Aspects of Computational Linguistics*, 68-95.Springer.
Stabler, Edward. 2001. Minimalist grammars and recognition. In C. Rohrer, A. Rossdeutscher, and H. Kamp (eds.), *Linguistic Form and its Computation*, pgs 327-352.Stanford, CA: CSLI.
Stabler, Edward. 2006. Sidewards without copying. In P. Monachesi, G. Penn, G.Satta and S.Wintner (eds.), *Formal Grammar'06, Proceedings of the Conference*, pgs. 133-146. Stanford, CA: CSLI.
Stabler, Edward. 2010. Computational Perspectives on Minimalism. In Cedric Boeckx, (ed.) *Oxford Handbook of Minimalism*, pgs. 617-641. Oxford: Oxford University Press.
Uriagereka, Juan. 1999. Multiple Spell-Out. In Samuel D. Epstein and Norbert Hornstein (eds.), *Working Minimalism*, pgs. 251-160. Cambridge, MA: MIT Press.