Labeling for Linearization

Cao Yu
yu.cao.nj@rutgers.edu

Abstract

A syntactic tree has been conceptualized minimally to be a set of nodes with a dominance relationship defined on them (Partee et al. 1990). But the syntactic objects constructed by Set Merge (Chomsky 2001 et seq) fit much closer with trees recursively defined in computer science. In this paper I consider how syntactic trees as such can be recursively linearized with tree traversal/walk algorithm familiar in computer science. I show that for tree walk to be applicable to an inherently unordered syntactic tree, an asymmetry has to be established between the two subtrees of a parent tree, so that the latter may be unambiguously co-labeled with exactly one of its subtrees. In modifying Chomsky's (2013, 2015) labeling algorithm to serve the purpose, I discuss the implications the label-based linearization theory has for the labeling algorithm itself and for the syntactic phenomena it has been used to account for. The theory thus provides a direct support in favor of the Strong Minimalist Thesis (Chomsky 2001 et seq) that the properties of syntactic computation are largely determined by interface conditions.

1 Introduction

When talking about the labels in phrase markers, we may identify two closely related questions: how do they come about (the origination, presupposing when)? What are they used for (the purpose)? Ever since X-bar theory recaptures the Bloomfieldian endocentricity generalization that disperses over various phrase structure rules (see Corver 2013; Fukui 2003 for a review), the *Projection Principle* and its descendants calculate the labels of nonterminals/parents from those of their children, the "nodes" they immediately dominate, upon identifying one of them as a *Head*. The labels of terminal nodes, i.e. atoms that enter computation, are provided by their lexical entries. Leaving aside their implementation details, what seems common to projection theories as such is that labeling goes hand in hand with structure building, so that (projected) labels of syntactic objects are constantly available throughout the computation, serving a purpose like matching the *subcategorization frames* (Chomsky 1965), constituting the *probe* and, as I understand, falling *goal* to the latter under *Agree* (Chomsky 2001 et seq), and being selected by external and internal *Merge* (Chomsky 2004, 2008).

But the *Strong Minimalist Thesis* (SMT; Chomsky 2001 et seq) leads us to wonder whether a said purpose of labels is itself justifiable, and if yes, whether labels are necessary for that purpose. For the examples cited above, the idea underlying subcategorization—satisfying selection properties even as building the syntactic objects—has gone away with the decisiveness assumption about computation (see Chomsky 2004 for discussion). Agree as a mechanism for resolving unvalued features is needed in the current Minimalist framework, as much as unvalued features themselves. But it is not clear why a probe has to look for the matching feature F in a projected label rather than the atom which carries F natively and from which F percolates/projects, if that atom is not shielded in a *phase*. One probable reason for assuming so is the idea that when Agree triggers displacement it is the syntactic object bearing the goal label that moves (Chomsky 2001). Insofar as internal Merge applies freely as external Merge does (Chomsky 2015), however, it would be more reasonable to purge Agree of correlating displacement and the concomitant decisiveness residual, and search elsewhere the reason why moving an atomic goal to the probe crashes computation. Similarly, Merge is of course indispensable, and it is of necessity for Merge to have some means to refer to the non-atomic syntactic object it selects. But it is another question what that means is or whether it is a projected label in the sense noted above.

In his recent works Chomsky (2013, 2015) proposes an algorithm that labels the syntactic objects larger than an atom at the phase level for (semantic) interpretation purpose. In the proposal's answer to the origination question, the timing of labeling echoes the doubts over uses of labels like those above: if projected labels are not there until a certain point in phase transition, any computational reference to them before that point would be impossible. This is the case for pre-phasal Merge, and also for phasal but pre-labeling operations like Agree (to be discussed later).

The proposal's answer to the purpose question fits the timing. But what a role syntactic labels plays in semantic interpretation depends largely on the syntax-semantics interface theory one adopts. It is not unusual for a semantic framework in the generative tradition to rely on little, if not none of the information encoded in the projected labels (e.g. Heim and Kratzer 1998); it is the lexical semantics of atoms that drives composition. The question remains open whether labels serve as a heuristic that reports crash based on label mismatch before semantic calculation detects the error. Inasmuch as the purpose of semantic interpretation itself is concerned, however, we may note it does not necessitate labeling *at* the phase level. It requires only that be done *by* the time *Transfer* concludes a phase (Chomsky 2004 et seq). This means that Merge with a built-in labeling effect can in principle serve the purpose equally well, as has been assumed since Chomsky 1995:chap. 4.

In this paper I will pursue a purpose of labeling that fits and arguably necessitates its phase-concluding timing: what if labels are added by Transfer in mapping the syntactic objects to the phonological component Φ, so that they can be linearized there and receive interpretation at the sensori-motor (SM) interface? This is a move to unite labeling and linearization, with the former being the means to achieve the latter. If workable, it would have implications for the labeling algorithm itself, and for the accounts of the syntactic processes that seem to enable the algorithm to apply. Below Section 2 discusses why labels are crucial for tree linearization, and what kind of labels a linearization algorithm for syntactic trees looks for. Section 3 modifies Chomsky's (2013, 2015) labeling algorithm and applies the results to some of the relevant examples discussed there. After a quick reflection on how the theory developed here might handle some of the most basic issues about cross-linguistic word-order variation, Section 4 concludes the paper.

2 How to Walk through a Tree

I start with a background setting which might seem distant at first sight, but whose relevance soon reveals itself. In a typical data structure textbook of computer science, binary trees are by default understood to be ordered, recursively defined as such (see Cormen et al. 2009; Wirth 1976; what follows draws heavily on sec. 4.4 of the latter):

(1) A *binary tree* is either

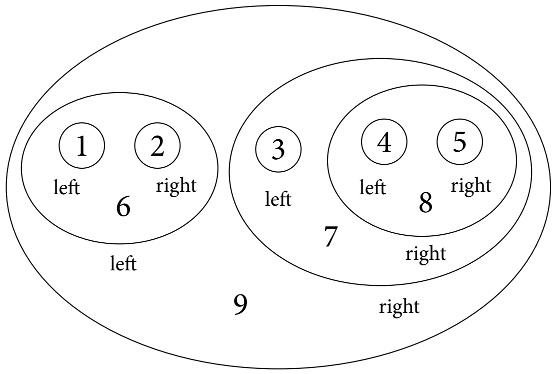
a. empty, or

b. a set consisting of a *root* node, and two disjoint binary trees called its *left subtree* and *right subtree*.

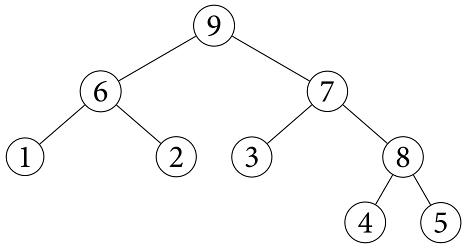
A binary tree is called *ordered* because there exists a fixed order between the two distinctly labeled subtrees for all roots, so that when we have to access the two subtrees of a root one after another, we know which comes first.

A binary tree defined by (1) can be represented with a diagram that directly reflects its set-theoretical essence (see Wirth 1976 for discussion on tree representations). (2a) shows an example, where roots are numbers 1–9, and empty sets are omitted. Equivalently, the tree structure in (2a) can be represented with roots alone, in a familiar graph structure that indicates the branching relationship. This is shown in (2b), with typographical left and right taking the jobs of labels.

(2) a.



b.



A primitive task on binary trees, called *tree traversal* or *tree walk*, is visiting all the nodes in a systematic way. The recursive structure of binary trees allows us to do so conveniently by recursion. With (2b) in mind, to walk through the tree rooted with 9, it suffices to visit root 9, and walk through its two subtrees in the same way. From this description emerges the algorithm *preorder walk*:

(3) PREORDER-WALK(x), where x is the root of a nonempty binary tree:

```
1 VISIT($x$)
2 **if** left subtree of $x \neq \emptyset$ **then** PREORDER-WALK(left subtree of $x$)
3 **if** right subtree of $x \neq \emptyset$ **then** PREORDER-WALK(right subtree of $x$)
```

(3) is called preorder in that VISIT of a root (line 1) precedes PREORDER-WALK of both subtrees, if they are nonempty (lines 2–3). Rearranging line 1 in between line 2 and line 3 yields *inorder walk*, and rearranging line 1 after both lines 2-3 yields *postorder walk* (though the name PREORDER-WALK no longer fits). Crucially, Wirth points out, in each case the individual nodes are visited in a specific sequential order, which means if we keep track of that, each walking algorithm above

actually *linearizes* tree (2b) in its own way. These results are shown in (4), with terminal nodes in boldface.

(4) a. *Preorder walk:* 9⁀6⁀**1**⁀**2**⁀7⁀**3**⁀8⁀**4**⁀**5**
b. *Inorder walk:* **1**⁀6⁀**2**⁀9⁀**3**⁀7⁀**4**⁀8⁀**5**
c. *Postorder walk:* **1**⁀**2**⁀6⁀**3**⁀**4**⁀**5**⁀8⁀7⁀9

(4a-c) display an important property of a recursive tree walk: a tree as a whole is linearized to a contiguous string. From this follows naturally the *Non-tangling Condition* (Partee et al. 1990), reinterpreted here as a condition on linearization of trees, not trees themselves. Limiting our interest to terminal nodes, we further see that the three walking algorithms give the same result: 1⁀2⁀3⁀4⁀5. This is because they all share the critical ordering of walking the left subtree *before* walking the right subtree, or in terms of the orderedness of the tree structure, label "left" is ordered before label "right". One may check, with that order reversed, all three algorithms uniformly give the reversed ordering of terminals node: 5⁀4⁀3⁀2⁀1.

From the presentation so far arise two observations. First, a recursively defined tree structure can be most naturally linearized with a recursive procedure, which has to make reference to the order among distinctly labeled substructures, or as formulated above, among labels themselves. Second, to within the arrangement of terminal nodes, that order alone completely defines the result of recursive linearization. In brief, if we understand tree linearization recursively, labels and their order are all that matters.

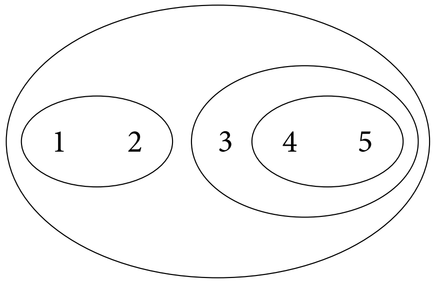
With these in mind let us turn to syntactic trees (= syntactic objects) constructed by Set Merge (Chomsky 2001 et seq). (Set) Merge takes two syntactic objects a and b and forms a set $\{a, b\}$ out of them, in turn a syntactic object. Thus iterative applications of Merge construct syntactic trees that can be recursively defined as such, comparable with (1):

(5) A *syntactic tree* is either

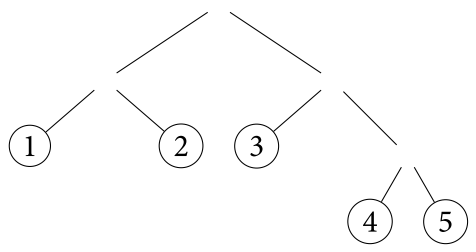
a. an atom, or
b. a set consisting of two disjoint syntactic trees called *subtrees* (with disjointness referring to *copies*), which are *siblings* of each other.

Definition (5) rejects an empty structure. Instead, it admits a single atom as a *degenerate* syntactic tree, which entails zero application of Merge. Unlike binary trees defined by (1), the substructures of syntactic trees defined by (5) are not automatically labeled, and given the property of sets, intrinsically not ordered. A comparable example to (2), with the same terminal nodes and grouping, is shown in (6a,b). The typographical left and right in (6b) should not be taken seriously.

(6) a.



b.



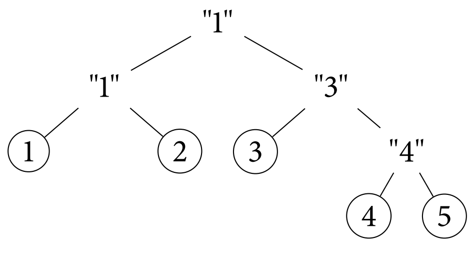
How can a tree like this be recursively linearized? One solution is labeling the non-atomic objects in such a way that not only can we distinguish the subtrees but also read off their order from labels, so that the order between labels need not be independently stated (e.g. *Laws of Precedence* in Fox and Pesetsky 2005). Ideally, the labels should come exclusively from the atoms that enter computation (the *Inclusiveness Condition*, Chomsky 1995:chap. 3). The long existing idea of projection provides precisely the solution we need. Consider a non-atomic syntactic tree, say, $c = \{a, b\}$, where a and b may or may not be atoms, and are asymmetrical and thus distinguishable in some dimension. Suppose a projects, in the sense that either c is labeled as "a" when a is an atom, or c inherits the label of a when a is not, then we say a and c are *co-labeled*.[1] Co-labeling gives us a natural order: if c and a are co-labeled, then a is ordered before b; otherwise if c and b are co-labeled, the reversed order obtains. Thus aside from trivial tree walk for an atom, where we simply visit the atom, to walk through a non-atomic syntactic tree like (6b), all we need to do is walk through its subtrees in the order determined by co-labeling. This gives us the following algorithm:

(7) Walk(x), where x is a syntactic tree:

```
1 if $x$ is an atom then
2     Visit($x$)
3 else
4     Let $y$ refer to $x$'s co-lableing subtree, $z$ the other subtree
5     Walk($y$)
6     Walk($z$)
```

As an example, if tree (6b) is labeled as in (8), then Walk(x), where x refers to the largest tree labeled "1", linearizes x as 1⌢2⌢3⌢4⌢5.

(8)



For algorithm (7) to be applicable, all non-atomic syntactic objects/trees must be labeled, and the way they are labeled should guarantee that a co-labeling relationship exists between each non-atomic tree and exactly one of its subtrees, which are asymmetrical in some dimension. Therefore the recursive linearization procedure, largely motivated by computational necessity, has posed

[1]In the next section, projection will be generalized to a feature F of a, in which case c is labeled as "F". Co-labeling is correspondingly generalized to hold between a[F] and "F".

a clear requirement on the labels it looks for. Let us now turn to Chomsky's (2013, 2015) labeling algorithm and see how it fits into the picture.

3 The Labeling Algorithm

Discussion in this section is limited to English. Chomsky (2013, 2015) has talked about how labels are projected in several different cases, depending on the syntactic properties of the elements involved:

(9) a. c = {X, YP}, X projects
b. c = {XP, YP}, feature F projects
c. c = {~~A~~, B}, B projects
d. c = {Z, B}, B projects

Chomsky's original ideas are summarized as follows. (9a) is the canonical case of label projection, involving a head X, presumably an atom, and a non-atomic element YP. In this case, it is X that projects. (9b) involves two non-heads, XP and YP. In this case projection is conditioned by the agreement relationship between XP and YP; if XP and YP agree on their "prominent" feature F, then F projects, according to Chomsky 2013. In practice, the prominent feature F is taken to be ϕ when YP = TP, and Q when YP = CP. (9c) involves a lower copy of a moved element ~~A~~ (striking out indicates that upon internal Merge, the phonological features of the lower copy are deleted; see Chomsky 2001), and another regular element B. Both ~~A~~ and B can be a head or a non-head. (9d) involves regular element B, a head or a non-head, and a deficient head Z, like a category-neutral root (Marantz 1997), morphologically weak T in English, Conj of a conjunction phrase etc.

Let us go through (9a-d) in more details to see how they address the unambiguous co-labeling relationship required by linearization. (9a) is as expected: the categorial asymmetry between a head and a non-head is well tended. With X being a head not among the deficient ones included in (9d), co-labeling c with X derives the head-initial order in English.

(9b) is subtle: for concreteness, let us consider Spec-TP and TP. Projecting ϕ does not give us the desired result, if unvalued ϕ of TP becomes indistinguishable from ϕ of Spec-TP under Agree (with the premise that labeling applies at the very end of phase transition, as assumed by Chomsky and us). This is because projecting ϕ in this case results in ambiguous co-labeling. To derive the actual Spell-Out in English where Spec-XP precedes XP, the ϕ of Spec-TP should project and be distinct from ϕ of TP, even after Agree. Indeed, while there is no (un)interpretability other than (un)valuedness, to make sure upon Transfer sending the syntactic object to the semantic component Σ, it is the redundant ϕ of TP, not the meaningful ϕ of Spec-TP that deletes, they should be somehow distinguishable. If so, Agree cannot completely eliminate their distinctions. Let us assume that upon valuation of an originally unvalued feature F, Agree marks F as "to be deleted on mapping to Σ", denoted as "dF", maintaining its distinction from the valuer F. Marking for deletion then fulfills two jobs: it allows transferring from narrow syntax to Σ to identify the redundancy for deletion, and it ensures unambiguous co-labeling in transferring from narrow syntax to the phonological component Φ. Thus we may maintain the formulation that for {XP, YP} where XP and YP agree on F, F projects, with the renewed import that it is the "originally interpretable F" that projects.

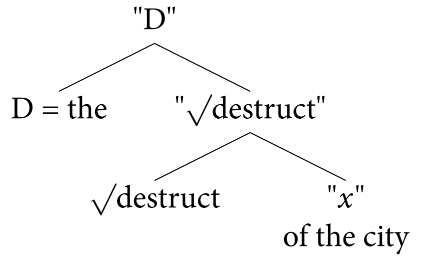
We have to pause for a moment to ask why in (9b) labeling has to refer to features. Chomsky suggests that labeling works like probing—essentially a *breadth-first search* strategy that exhausts nodes of the same depth before moving on to the next level of depth in search of a certain goal (see

Cormen et al. 2009). So that with c = {XP, YP} the labeling algorithm probing for heads finds X and Y. But what matters is whether the algorithm finds them at the same depth, and if yes, whether they are on a tie in terms of projecting ability. Taking the example of {Spec-TP, TP} again, we may imagine a case where Spec-TP has its own subject, i.e. Spec-TP = {Spec-DP, DP} (e.g. *John's brother*), in this case T stands out as the closest goal in terms of search depth. Resorting to head deficiency in (9d) does not help; in a simpler case where Spec-TP = {D, NP}, D and T are found at the same depth, but if T is unable to project for its weakness, then D stands out as the closest adequate goal. Thus no matter English T projects or not, if labeling works like probing, there is always a case where reference to features is not necessary. What complicates the picture even more is that it is nontrivial that the closest adequate goal X found in the XP branch necessarily co-labels with XP (e.g. XP = {Spec-TP = {Spec-DP, DP}, TP}; assuming adequacy of T and deficiency of Y). When they do not, labeling c = {XP, YP} as "X" brings neither XP nor YP into co-labeling relationship with c. These complexities can be avoided and the necessity of reference to features can be strengthened if labeling does not work like probing. Instead, the asymmetry it looks for can be very local: for a syntactic tree $c = \{a, b\}$, the asymmetry between a and b in some dimension should suffice. If categorial asymmetry cannot be found as for a = XP and b = YP, then the "prominent" feature F on which they agree rises up, as one of them bears F and the other dF in discussion above.

With this understanding, (9c) is as expected: the phonological asymmetry between ~~A~~ and B makes a plain distinction between the two. If co-labeling relates directly to linearization precedence, it is intuitive that B, the element that retains its phonological feature, is in a better position to project. (Otherwise why should something that cannot be heard compete for Spell-Out priority?) The account of B's priority provided by Chomsky is reasonable, but not available to us—that a lower copy is not a proper goal for probing, since we have seen it can be problematic to make labeling rely on probing.

In (9d) co-labeling certainly obtains between c and B. But the claimed asymmetry between Z and B is describing the result, that B projects but Z does not. Given the theory developed here, we therefore need to verify to what extent the said inability of a head to project makes correct predictions about word order. Let us first discuss category-neutral roots, which naturally introduces discussion of T. Assuming with Marantz (1997) that syntactically D functions like v in its ability to type a root (D being one of many flavors of little n), *the destruction of the city* would be represented as follows:

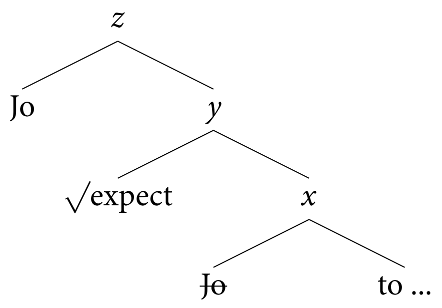
(10)



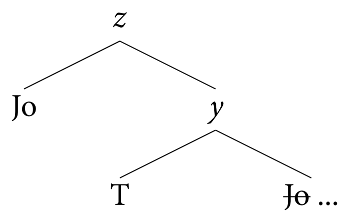
If root $\sqrt{}$*destruct* projects like a regular head, as the labels above indicated, tree walk (7) would linearizes (10) as expected. Otherwise projecting "x" gives the wrong order. Along side the positive example (10) showing a root does project, we also need to examine the example (11a) Chomsky provides to show the opposite, which involves the Exception Case Marking construction (ECM). The idea is that the structure of (11a) parallels that of (11b), so that if T fails to label y in (11b) for its weakness and thereby necessitates subject raising, then it can be inferred from object-raising in

(11a) that root $\sqrt{}$*expect* is weak as well.

(11) a.



b.



There are several problems we may note in this inference. As the inference is based on an analogy, let us consider the deficiency of T in (11b) before (11a). From discussion of (9b) above we have seen that in (11b) for z = {*Jo*, y} to be labeled, y has to be labeled by T, which projects its feature bundle that includes dϕ, so that on absence of the categorial asymmetry between y and *Jo* (assuming with Chomsky that nominal expression *Jo* is non-atomic), the valuation asymmetry between them promotes ϕ of *Jo* as the label of z. That English T has to project can also be seen from examples where it has overt realization like *will*, which is to be linearized before its verbal complement, suggesting that {T = *will*, XP} receives its label from T. Making T a projectable head does not necessarily lose us an account of the basic EPP effect. Chomsky has pointed out that in case of transitive sentences, subject-raising is motivated by the labeling requirement, namely, to label c = {DP = Spec-vP, vP} without satisfying the agreement condition on the two elements, DP = Spec-vP has to raise to create a situation of (9c). In case of unaccusative sentences, after having T's unvalued ϕ resolved under Agree, that ϕ receives a marker for deletion that is useful in mapping to semantic component Σ, but useful for mapping to phonological component Φ only if a sibling subject carrying the same ϕ causes co-labeling ambiguity. Suppose there is an operation in Transfer that applies *before* sending syntactic objects to Σ and Φ, and that blindly removes disambiguation markers useless for the labeling procedure, then this mechanism simplifies feature representation in mapping to Φ but bleeds the redundancy identification in mapping to Σ. Therefore, the problem of unaccusative sentences without having the internal argument raised to the subject position or having that position filled by an expletive reduces to the problem of forcing Σ to interpret "redundancy".[2] Thus T itself does not make a good example of an unprojectable head.

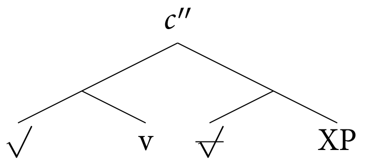
We still have to consider if it is not the weakness of T or V (a root) that triggers nominal raising, what should be responsible for that. Indeed, what triggers object-raising in (11a) is not the labeling failure of y, but rather that of x. We may substitute x for a finite CP, e.g. *We expected that Jo would come*, in which case there is no reason to assume that subject-to-object raising actually happens, but it can be shown that there is no labeling failure either. The reason is that in a finite clause, c = {Spec-TP, TP} can be labeled by feature projection upon agreement. But the same might not extend to an infinitival clause: suppose an infinitival T, realized as *to*, does not carry the needed unvalued ϕ. Then in this familiar agreement-absent situation, to label c Spec-TP has to raise to create a situation of (9c).[3]

[2]This term is more damaging than its innocent look suggests. ϕ is to be associated with individuals (probably as their modifier), not event modifiers that convey information about tense. If a feature ultimately translates into expressions in semantic calculus, "redundancy" actually entails composition crash.

[3]Nonetheless, the current story has to assume with Chomsky, as I understand, that Spec-VP and VP headed by an ECM verb enter agreement.

Besides the ECM-EPP analogy, another reason that might motivate weakness of root is the labeling problem for c = {X, R} where X is a categorizer and R a root. This can be seen as an atomic analogy of (9b). While labeling by agreement does not obtain, otherwise motivated head-to-head movement allows us to get around the labeling difficulty by evacuation, without assuming deficiency of roots. Using *pair Merge* strategy designed for adjoining structure (Chomsky 2004, 2013), we derive c' ={<R, X>, ~~R~~}. If only the first coordinate of a pair is visible to the labeling algorithm as Chomsky suggests, then a pair of heads is recognized by the labeling algorithm as a head, namely c' will be labeled as "R". This should apply before flattening <R, X> to {R, X} (*Simplify* in Chomsky 2004), otherwise with the transitive verb in (12),

(12)



c'' composed of two disagreeing non-atoms cannot be labeled, if evacuation is unavailable. It is reasonable to build flattening of pairs into our tree walk algorithm, since the order is already encoded in a pair <A, B>, the algorithm does not need otherwise calculated labels to determine the order between A and B: flattening <A, B> to c = {A, B} automatically labels c as "A". This entails that if the second coordinate B is non-atomic, B and the subtrees contained in B should all be properly labeled *before* it pair Merges with A, since B is invisible to the labeling cycle pertinent to A. Given that labeling applies as part of Transfer, it follows in turn that B constitutes a phase. The conclusion suggests an interesting way to derive the island effects of adjuncts, which I leave open here. Thus with head movement and otherwise needed pair Merge and pair flattening, we can dispense with weak root assumption in accounting for the last piece of evidence seemingly in favor of that.

One more item on the list of deficient Z in (9d) is Conj. One of the reasons that T projects applies here as well: we know that syntactic object c = {Conj = *and*, B} should be linearized as *and*⌢B, but this cannot be done without projecting Conj. The puzzle left open in Chomsky 2013 is the labeling of c' = {A, c = {Conj, B}}, where A is non-atomic. A conjunction structure functions like either of its conjuncts, as Chomsky remarks, which gives us plenty of reasons to speculate that c' co-labels with A (or B). But it is not obvious there should be feature agreement between A and c if we try labeling by agreement ((9b)), nor is it obvious that there is otherwise motivated movement for us to try try labeling by evacuation ((9c)). Stipulating deficiency of Conj does not help, since it forces co-labeling B with c (which should be avoided in the current story), but does not explain why A can project upon absence of categorical, valuation, and phonological asymmetries. A possible solution to the puzzle is provided by pair Merge, as Chomsky (2013) suggests for "unstructured coordination": a conjunction structure c' functions like A merely because computation sees only A. Therefore the coordination structure intended by c' is actually c'' = <A, c = {Conj, B}>, which will be automatically flattened to c' with label "A" by tree walk along the refinement in the last paragraph. And given the conclusion reached in the last paragraph, this implies c, a non-atom, should have been labeled when it Mergers with A, and thus Conj might well be a phase head. This suggests an interesting way to derive the island effects of coordination (*Coordinate Structure Constraint*; Ross 1967), though unfortunately for the second conjunct only. I leave the problem open again.

Summarizing the case-by-case discussion of Chomsky's labeling algorithm in (9a-d) and modifications introduced along our discussion, we establish the following results:

1. The unambiguous co-labeling relationship required by tree walk/linearization provides a reliable principle for labeling.

2. Labeling does not probe. It reduces to projection, based on three local asymmetries:

i. Phonological asymmetry:	✓Pronounceable vs. ×Unpronounceable	{A, ~~B~~}
ii. Categorial asymmetry:	✓Atom vs. ×Non-atom	{X, YP}
iii. Valuation asymmetry:	✓Valued in lexicon vs. ×Valued in syntax (by Agree)	{F, dF}

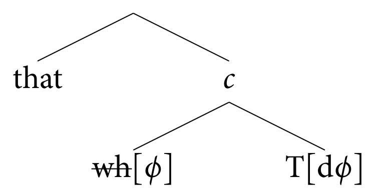
3. Mark for deletion (of redundant features) applies blindly under Agree. When the difficulty in creating (iii) does not arise, its effects are undone.

4. A pair resulting from pair Merge is flattened by tree walk. The second coordinate of a pair has to be labeled beforehand (assuming atoms are trivially labeled).

By sticking strictly to the first and second points above, we have seen that the projection inability assumption about certain English heads, including at least category-neutral roots, T, and Conj, can be abandoned. It remains open whether there exist heads that are genuinely unable to project. It also remains open whether there is any additional asymmetry apart from the ones listed in (i)-(iii), or whether (i)-(iii) can be reduced to fewer more basic asymmetries.

So far we have been assuming (i)-(iii) may function independently, but from the discussion of (9d) above, we have seen a case that involves contradictory instructions given by (i) and (ii). In (12), after head-moving the root to its categorizer, we derive a tree of the form c = {~~R~~, XP}, where ~~R~~ is the lower copy of the root. If V-to-v movement happens regularly in English, there should be no difficulty in labeling c. Thus one of their instructions has to give in, but we cannot tell which from what can be heard. For the reason mentioned in discussion about (9c), let us assume it is the instruction given by phonological asymmetry that prevails. A critical example showing the interaction between (i) and (iii) involves the *that*-trace effect:

(13)



As Chomsky suggests, if *that*-trace effect reduces to labeling failure of c in (13), then we may conclude that neither of the instructions given by (i) and (iii) surrenders. The conclusion is compatible with an independently motivated assumption made in our discussion of the ECM effect in (11a), that infinitival T does not carry an unvalued ϕ; otherwise labeling failure of x would arise there. I save discussion about why (i)-(iii) should interact this way to the next section.[4]

We now have a labeling algorithm that tells us the label of $c = \{a, b\}$ based on the kind of asymmetry between a and b. But with a given non-atomic syntactic tree x, we cannot label the non-atomic subtrees in x or x itself in any random order. This is because whenever the projecting element is non-atomic, it is its label that projects, which means a and b have to be labeled before c can be

[4]The interaction between (ii) and (iii) is rare, as (iii) deriving from (9b) presumes category symmetry. But below we will see example (20) suggesting checking valuation asymmetry only if categorial asymmetry is absent.

labeled. Moreover, asymmetry (iii) directly refers to the labels of the substructures involved.[5] While the situation initially suggests a bottom up labeling procedure, as we will do manually below, it can be expressed more conveniently with a recursive algorithm, to make best use of the recursiveness of syntactic trees. We know how to label an atom, which is the label of itself. To label a non-atomic tree, all we need to do is applying the same recursive labeling procedure to its subtrees, order irrelevant, and feed the labels to one of the projection options (i)-(iii). To avoid confusion that has not arisen until this point, let us rename what we have called "labeling algorithm" as "projection algorithm", reformulated in (14a), and save "labeling algorithm" for (14b):

(14) a. Project(a, b), which returns a label for trees labeled by a and b (order irrelevant):

```
1  Let x refer to the tree labeled by a, y the tree labeled by b
2  if x and y are phonologically asymmetrical then
3      Let z refer to the pronounceable one of x and y
4      if z is lexically valued then return z's label
5  else if x and y are categorially asymmetrical then
6      Let z refer to the atomic one of x and y
7      return z's label
8  else if x and y are valuation-wise asymmetrical then
9      Let z refer to the lexcially valued one of x and y
10     if z is pronounceable then return z's label
```

b. Label(x), which labels a given syntactic tree x:

```
1  if x is an atom then
2      Label x as x
3  else
4      Let y refer to one subtree of x, z the other subtree
5      Label(y)
6      Label(z)
7      Label x as Project(y's label, z's label)
```

(14b) calls (14a) in its execution. If none of the conditional branches applies, or if the phonological asymmetry and valuation asymmetry disagree, the result of function Project is undefined, which causes computation crash.

We also need to refine our tree walk algorithm, so that it can linearize pairs resulting from pair Merge. To improve computation efficiency we may further cut off phonologically empty branches from tree walk, as there is no need to linearize unpronounceable elements. Therefore (7) is updated as follows, where lines 3–5 flatten a pair, and newly introduced pronounceability condition in line 6 avoids walking into an unpronounceable tree.

[5] Following Chomsky 2013, the idea is that the label of $c = \{a, b\}$ stores the features of the element, say a, that projects.

(15) Walk(x), where x is a syntactic tree:

```
if $x$ is atomic and not a pair then
    Visit($x$)
else if $x$ is of the form $<y, z>$ then
    Let $x'$ be $\{y, z\}$ with label $y$
    Walk($x'$)
else if $x$ is pronounceable then
    Let $y$ refer to $x$'s co-lableing subtree, $z$ the other subtree
    Walk($y$)
    Walk($z$)
```

Having worked out the projecting and labeling algorithms that feed the tree walk algorithm, let us now try to run them over some concrete examples, including a regular SVO sentence, *wh*-movement that shows *that*-trace effect, and *wh*-movement in an ECM construction. The presentation reflects the results one would get from a step-by-step execution of the algorithms. For simplicity, let us regard a nominal expression as a chunk, without considering its internal labeling/linearization. Also let us assume that C and v alone (on the verbal spine) are phase heads, and restrict a *phase* to

(16) a syntactic tree composed of a *phase head* and its sibling called *phase head sibling*,

thereby reinterpret *Phase Impenetrability Condition* (Chomsky 2001 et seq) as limiting phase-external access to the interior of a phase head sibling to the subtrees of a phase edge:

(17) A *phase edge* is a syntactic tree composed of a non-atom and either

a. a phase, or

b. a phase edge.

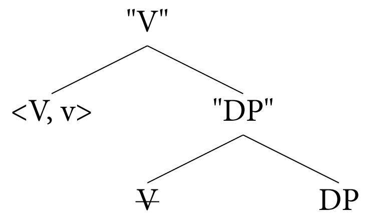
A phase edge defined as such is therefore not part of a phase. The narrowness of definitions (16) and (17) is merely for presentation purpose, whose implications are yet to be explored. With this understanding, let us assume, unconventionally, that Transfer covers and thus labels the *whole* phase. The Transfer coverage may extend to phase edges in certain situations. One is that the current phase is the last phase of derivation. For example, in a regular *wh*-question like *who did Jo see*, the last phase is c = {C, TP}, the final phase Transfer has to cover the phase edge c' = {*who*, c}.

In the first example (18a), the derivation up to v-phase is shown in (18b). Let us assume V that does not Merge with non-phase (e.g. TP of ECM) does not inherit phasehood from v,[6] and head-raising V to v still gives us a phase head <V, v> (*C-to-T and v*-to-V Inheritance*, Chomsky 2008). Thus when <V, v> triggers Transfer, by phonological asymmetry DP object projects its label, and then by categorial asymmetry the first coordinate V of <V, v> projects its label. Turning to (18c), by the time C triggers Transfer, v-phase projects by phonological asymmetry; T projects by categorial asymmetry; ϕ of DP projects by valuation asymmetry; finally C projects by categorial asymmetry. One may walk through the tree to see that these labels yield the correct linearization.

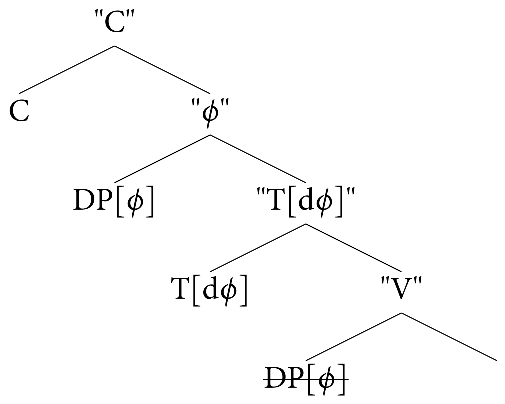
(18) a. Jo saw Ben.

[6] This suggests DP might be a phase.

b.



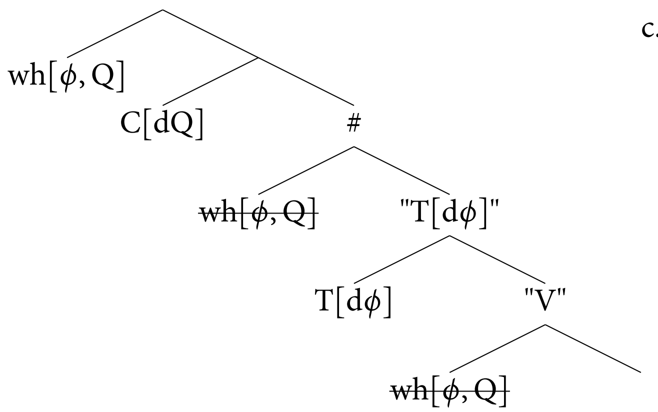
c.



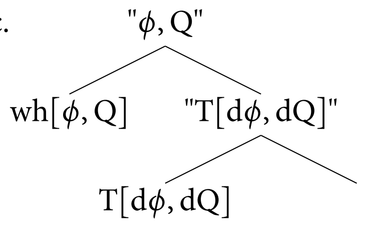
The second example elaborates *that*-trace effect we have seen in (13). The derivation up to the lower v-phase (19a) is similar as above. By the time C = *that* triggers Transfer, as shown as (19b), v-phase projects by phonological asymmetry; T projects by categorial asymmetry; but then for the tree composed of "Spec-TP" and "TP", phonological asymmetry conflicts with valuation asymmetry. A sharp sign indicates a labeling failure. On the other hand, if we assume with Chomsky (2015) that right before C undergoes deletion, T inherits phasehood and other features from C and triggers Transfer, the labeling failure disappears in (19c). Here the new phase edge cannot be left to be labeled by the next phase; otherwise from successive movement of *wh*-phrase the failure in (18b) would arise again. Thus we identify another case of Transfer coverage extension: upon phasehood inheritance.

(19) a. Who do you think (*that) __ saw Ben?

b.



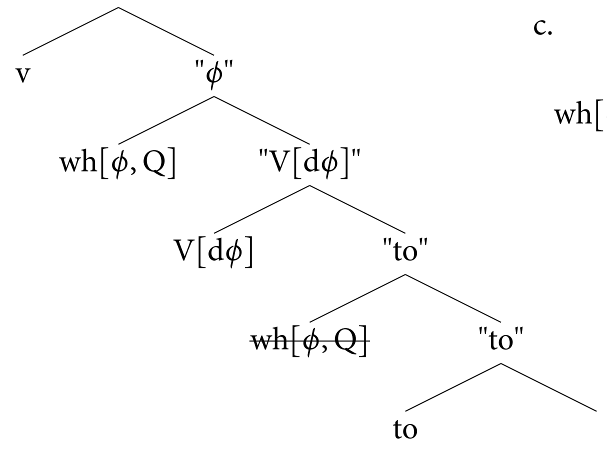
c.



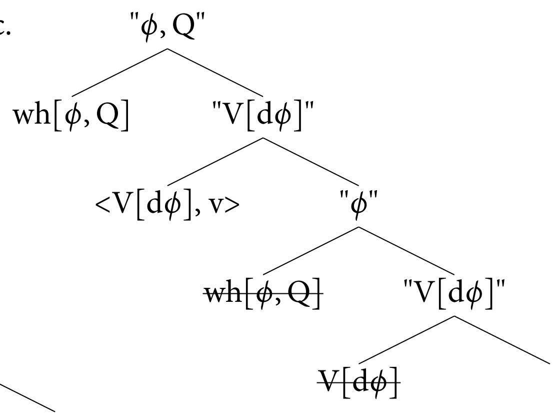
The third/last example (20a) shows a *wh*-analogy of (11a). Omitting derivation below T = *to*, when *wh*-phrase reaches "Spec-VP", it has to seize the last chance to project its feature label, otherwise the projection conflict in (19b) would arise again. This suggests phasehood inheritance happens between v and V that Merges with a non-phase. Thus when V triggers Transfer in (20b), whose coverage now extends to the phase edge as said before, T = *to* projects by categorial asymmetry; TP projects by phonological asymmetry; V projects by categorial asymmetry; ϕ of *wh*-phrase projects by valuation asymmetry. After internal Merging the *wh*-phrase and V, V as the first coordinate triggers Transfer with an extended coverage again, as shown in (20c) . V projects by categorial asymmetry (see fn. 4 and lines 5–8 of projection algorithm (14a)), and the features of the *wh*-phrase project by valuation asymmetry. For discussion about V-T analogy, that V may carry a ϕ feature, and operation ordering which is crucial for labeling here, see Chomsky 2015.

(20) a. Who do you expect __ to see Ben?

b.



c.



Along with other details of the theory developed here, the modified projection algorithm performs as expected for the testing examples above. Its full strengths and of course weakness await future investigations.

Let us note one more question before concluding this section. We have been assuming labeling algorithm (14b) applies as part of Transfer, on way of mapping syntactic objects to the phonological component Φ. But when does linearization algorithm (15) apply? Abstracting away from phonological processes, if by linearizing syntactic tree x we mean all the atoms in x, minus those in the branches cut off (see line 5 of (15)), are fixed to a *sequential* structure, then it is evident that later derivation cannot alter that atom sequence. The consequence of displacement is unpredictable, since there is no more hierarchies to tell whether a certain operation observes phase impenetrability—notions like siblings and subtrees defined on syntactic trees are undefined for sequences. (20c) shows extraction from a Transferred domain is possible or perhaps even necessary. It then follows that while labeling applies phase by phase, linearization cannot follow too closely. The question remains open whether there is an overall linearization, or it applies multiple times.

4 Conclusion

I have pursued the linearization purpose of syntactic labels and some of its implications for the syntactic theory we have. From the theory developed here we have seen to what extent conditions imposed on the interface (in this case, sensori-motor interface) and computation necessity may have a say in how languages should be structured. It therefore provides immediate support for the direction of the broader research program the Strong Minimalist Thesis (Chomsky 2001 et seq) is heading to.

Since labels are no more than projections of computation atoms, they provide a readily available means to order by-definition unordered structures like syntactic trees. Tree walk as a recursive algorithm emerges naturally from the recursiveness of syntactic trees themselves, so that it straightforwardly captures the correspondence between hierarchical asymmetry and precedence asymmetry in natural language linearization, as has been highlighted in Kayne's (1994) approach. It also confirms Kayne's hypothesis, from another angle, that a syntactic tree of more than two branches is impossible for being not able to be linearized: unambiguous co-labeling distinguishes only two elements. We may note that it is even computation-wise more efficient than Kayne's approach. For a syntactic tree with n atoms and thus $2n - 1$ nodes in total (a property of *full binary trees*; see Cormen et al. 2009), pairwise c-command checking yields an algorithm of a quadratic time complexity (roughly $(2n-1)^2/2$), whereas tree walk can linearize a tree with a linear time complexity $(2n-1)$, since each node is visited only once.

In this concluding section, let us briefly consider how a label-based linearization theory should address some of the most basic aspects of cross-linguistic word order variation. Tree walk reads off orders from unambiguous co-labeling. This statement indicates two places where word-order variation might reside. One has to do with how tree walk/the linearization algorithm *interprets* co-labeling, i.e. does the co-labeled or the otherwise labeled subtree come first? The other has to do with how the projection algorithm *decides* co-labeling, i.e. with x and y of asymmetry so-and-so, which one projects its label? It seems reasonable to consider only one of the two places, since systemically reversing every option in both brings us back to where we start. Suppose the projection algorithm like the one designed for English is universal, and it is the linearization algorithm that is subject to variation. But given that both verbs (for being atoms) and subjects (for being lexically valued) project in English, this entails a head-final language—one that orders hetero-labeled subtree first—would also be a Spec-final language. Mark Baker (p.c.) points out that head-final languages are way more common than Spec-final languages in typology. Therefore we may suspect that interpretation of co-labeling relationship is universal; it is the projection algorithm that varies across languages. We therefore look back to the projection algorithm in (14a).

Limiting ourselves to the three by now familiar asymmetries, i.e. the phonological asymmetry, the categorial asymmetry, and the valuation asymmetry, let us first consider the projection decisions made with respect to them *in their own*. Projecting the pronounceable element over the unpronounceable one seems a natural choice, but even if the preference is reversed, we cannot tell from what we hear: concatenating a string s with an empty string ε as either $s \frown \varepsilon$ or $\varepsilon \frown s$ makes no difference. Projecting the atom over the non-atom seems an arbitrary choice, which yields a head-initial language. The inverse seems equally likely, which yields a head-final language. Thus the projection decision for the categorial asymmetry interprets the traditional *Head Parameter* (Stowell 1981) in a new context. Projecting the lexically valued feature over the syntactically valued feature seems more arbitrary than natural, but to the extent that Spec-final is indeed a rare typological pattern, and the possibility exisits that this order in its exponent may receive alternative accounts, we may tentatively assume like the pronounceability preference, the preference for lexical valuation is also of a universal nature. In deriving EPP effects in English, I have posited a speculative mechanism that removes "markers for deletion" when the lack of subject excludes the co-labeling ambiguity. If it is the preference for projecting features that are lexically valued—a property computation system can tell only from the presence/absence of the said markers, then that mechanism might be more reasonable. Apart from this, we have noted in the previous section that when the phonological asymmetry and the categorial asymmetry conflict, the latter's instruction gives in, but when the phonological asymmetry and the valuation asymmetry conflicts, neither of their instructions compromise. This might be taken to indicate that the pronounceability preference and the lexical valuation preference form a natural class of a universal nature, to the exclusion of the atom preference. If so, we may partly explain the relative stability in Spec-directionality compared to the relative freedom in head-directionality.

The second dimension along which the projection algorithm may vary is suggested in the lines above: the interaction of the contradictory instructions given by different asymmetries. Besides the two interactions mentioned above, from the successive *wh*-movement in English ECM constructions ((20)) we have seen evidence in favor of the instruction given by categorial asymmetry over that given by valuation asymmetry, somewhat surprisingly. This would hold only if asymmetry interaction relationships are *not* transitive (i.e. from that categorial asymmetry prevails over valuation asymmetry and that pronounceability asymmetry prevails over categorial asymmetry, we cannot infer pronounceability asymmetry prevails over valuation asymmetry), thereby do not form a partial order, and thereby are not amenable to an Optimality Theory analysis (Prince and Smolensky 2004). But it is nonetheless programmable, as shown in (14a) for English. The empirical predictions about word-order made by variations along this dimension are much more nontrivial than those discussed above, opening a new window to think about otherwise mysterious phenomena in word-order variation.

References

Chomsky, Noam. 1965. *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.

Chomsky, Noam. 1995. *The minimalist program*. Cambridge, MA: MIT Press.

Chomsky, Noam. 2001. Derivation by phase. In *Ken Hale: A life in language*, ed. Michael Kenstowicz, 1–52. Cambridge MA: MIT Press.

Chomsky, Noam. 2004. Beyond explanatory adequacy. In *Structures and beyond: The cartography of syntactic structures*, ed. Adriana Belletti, volume 3, 104–131. Oxford University Press.

Chomsky, Noam. 2008. On phases. In *Foundational issues in linguistic theory: Essays in honor of Jean-Roger Vergnaud*, ed. Robert Freidin, Carlos P. Otero, and Maria Luisa Zubizarreta, 133–166. Cambridge, MA: MIT Press.

Chomsky, Noam. 2013. Problems of projection. *Lingua* 130:33–49.

Chomsky, Noam. 2015. Problems of projection: Extensions. In *Structures, strategies and beyond: Studies in honour of Adriana Belletti*, ed. Elisa Di Domenico, Cornelia Hamann, and Simona Matteini. John Benjamins Publishing Company.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to algorithms*, 3rd edition. Cambridge, MA: MIT Press.

Corver, Norbert. 2013. Lexical categories and (extended) projection. In *The Cambridge handbook of generative syntax*, ed. Marcel den Dikken, 353–424. Cambridge University Press.

Fox, Danny, and David Pesetsky. 2005. Cyclic linearization of syntactic structure. *Theoretical linguistics* 31, no. 1-2:1–45.

Fukui, Naoki. 2003. Phrase structure. In *The handbook of contemporary syntactic theory*, ed. Mark Baltin and Chris Collins, 374–406. Oxford: Blackwell.

Heim, Irene, and Angelika Kratzer. 1998. *Semantics in generative grammar*. Oxford: Blackwell.

Kayne, Richard S. 1994. *The antisymmetry of syntax*. MIT Press.

Marantz, Alec. 1997. No escape from syntax: Don't try morphological analysis in the privacy of your own lexicon. In *University of pennsylvania working papers in linguistics*, volume 4.2. University of Pennsylvania.

Partee, Barbara H., Alice ter Meulen, and Robert E. Wall. 1990. *Mathematical methods in linguistics*. Kluwer Academic Publishers.

Prince, Alan, and Paul Smolensky. 2004. *Optimality theory constraint interaction in generative grammar*. Blackwell Publishing.

Ross, John Robert. 1967. Constraints on variables in syntax. Doctoral Dissertation, MIT.

Stowell, Timothy Angus. 1981. Origins of phrase structure. Doctoral Dissertation, MIT.

Wirth, Niklaus. 1976. *Algorithms + data structures = programs*. NJ: Prentice-Hall.