A Computational Model of Language Generation Applied to English *Wh*-questions

Jason Ginsburg
Center for Language Research
University of Aizu
Aizuwakamatsu, Japan
jginsbur@gmail.com

***Abstract*—This paper presents a computational model of language generation, based on Phase Theory, that automatically constructs sentences from underlying numerations. This model incorporates explicit algorithms that determine selection and merger of Lexical Items from a subnumeration, determine the labels of Merged syntactic elements, account for movement of elements within a derivation, and account for when phrases are sent to Spell-Out. This paper shows how this model automatically produces the derivation of an English *wh*-question.**

***Keywords*-Phase Theory; *wh*-questions, computer modeling**

I. Introduction

In this paper, I describe an attempt to create a precise and unambiguous computational model of sentence generation based on Phase Theory ([1], [2], [3], [4]). The goal of this work is to model a version of Phase Theory on a computer and in the process create a more accurate theory of how the human mind generates sentences.

In Phase Theory, a sentence is constructed via a bottom-up process in which Lexical Items (LIs) are selected from a numeration, which consists of subnumerations, and are Merged together. A derivation is broken up into phases for which v* (transitive v) and C (also possibly D) are phase heads. A derivation is subject to the Phase Impenetrability Condition (PIC), which determines when phrases are sent to Spell-Out (or to Transfer). The PIC has been formulated in at least two ways (cf. [5]). According to one version ([1], [2]), in (1), when the phase head v* or C is Merged, the complement of the phase head, VP or TP, is sent to Spell-Out. According to the other version [3], in (1), when C is Merged, VP, which is the complement of v*, is sent to Spell-Out.

(1) $[_{CP}$ C $[_{TP}$ T $[_{vP*}$ v* $[_{VP}$ V ...]]

Once a phrase is sent to Spell-Out, its contents are no longer accessible to higher operations. Within Phase Theory, a head with an uninterpretable feature functions as a probe that Agrees with a goal (if present) that has a matching interpretable feature. Crucially, a probe cannot agree with a goal that is within a phrase that has been sent to Spell-Out.

There are some issues with Phase Theory that the research presented in this paper attempts to clarify. First, it is generally assumed that an LI selects for another LI; e.g., v* selects for V and V selects for DP, but it is not entirely clear how this selection process works. Also, issues arise with respect to the relationship between phases, subnumerations, and the timing of Spell-Out. A subnumeration can consist of LIs that form a phase, or it can consist of LIs that form a subject or adjunct, which must be formed outside the main spine of a derivation (see [6] for discussion of why this is the case). Under the PIC, a phase edge remains visible to a higher phase and a complement of a phase head is sent to Spell-Out. Thus, there is a lack of a one-to-one correspondence between sending an element off to Spell-Out and 'phase-hood'. Furthermore, the PIC creates problems for notions of movement. In cases of long distance movement in *wh*-questions, EPP features are required to bring a *wh*-phrase to its scope position. In the question 'What did you think that Bob ate?', 'what' must move through all intervening phase edges to arrive in its scope position, as shown in (2). To account for this fact, [1] suggests that a phase head can optionally have an EPP feature that attracts a *wh*-phrase. However, if a derivation proceeds in phases, and a lower phase is not 'aware' of the contents of a higher phase, then it is not clear how an intervening phase head can *know* that it requires an EPP feature (cf. [7]).

(2) $[_{CP}$ What $C_{[EPP|}$ did $[_{v*P}$ t_{what} $v^*_{[EPP|}$ you think $[_{CP}$ t_{what} that$_{[EPP|}$ Bob $[_{v*P}$ t_{what} $v^*_{[EPP|}$ t_{Bob} ate $t_{what}]]]]$

This paper describes a computational model of language generation that attempts to shed light on some of the complex issues and problems with Phase Theory. This model is novel in that it is, to my knowledge, the first attempt to create a computer program that, from the perspective of Phase Theory, automatically constructs interrogative constructions from underlying numerations. In the following sections, I describe the basic algorithms that this model uses and I demonstrate how the model successfully constructs the derivation of an English *wh*-question from a numeration.

II. How the model works

I created a computer program, implemented in the Python programming language, that is fed a numeration, from which it automatically constructs the derivation of a sentence. In this section, I explain the basic algorithms of this model.

The model incorporates a bare phrase structure (cf. [8]) view of a syntactic tree. A tree is formed from iterative Merge of two elements at a time, and the label of the tree is the label of one of the Merged elements; e.g., a phrase has the label D instead of DP.

A derivation begins with a numeration such as (3), which contains the LIs X and Y as well as another subnumeration that contains the LIs Z and W, and so on.

(3) {X, Y, {Z, W, {P, {Z, W}, Q}}}

Once fed a numeration, the model searches for the most embedded subnumeration by recursively applying (4).

(4) *Choose Subnumeration: For each element in S (a subnumeration), if S is a set S' (another subnumeration), select S'.*

Given the numeration (3), the Selector (the element of the model that selects elements from a numeration/subnumeration) will, via iterative application of (3), initially select the most embedded subnumeration {Z, W}.

Once a subnumeration is selected, LIs are selected and Merged. The features that LIs contain play an important role in this process.

As shown in Table 1, there is one type of unvalued feature (a), and there are two types of valued features (b-c). The ':_' in (a) represents an unvalued feature. An unvalued feature

	Feature	Description
(a)	Unvalued	Feat:_
(b)	Valued	+Feat:X
(c)	Valued	Feat:X

Table I
FEATURES

of type (a) in Table 1 must be valued by a matching valued feature. For example, T has unvalued phi-features 'Phi:_' that obtain a phi-feature value from a valued 'Phi:X'. This model also utilizes unvalued label features that are valued by the label of a Merged element. For example, an unvalued 'N:_' feature contained on a D is valued by the 'N' label of a noun complement. A valued feature of type (b) must undergo a feature checking relation in which the valued feature assigns an unvalued feature a value. The '+' in the '+Feat:X' signifies that the feature must be assigned. This type of feature functions as a probe that searches for a matching unvalued feature. For example, (structural) case is a feature of this type; a '+Case:X' feature that must assign a value to a matching unvalued 'Case:_' feature. On the other hand, the valued feature in (c) need not, but can, undergo a feature checking relation. In this model, an N comes with valued phi-features of type (c). These phi-features can value matching unvalued phi-features on T. However, there is no inherent need of the phi-features on N (unlike a case feature on T) to value a matching unvalued feature.

Following [9], features are also specified for interpretability. Interpretable features remain throughout a derivation and uninterpretable features must be eliminated for a derivation to converge. For example, phi-features on N are interpretable, represented as 'iPhi:X' ('i' signifies interpretable), and unvalued phi-features on T are uninterpretable, represented as 'uPhi:_' ('u' signifies uninterpretable), and must be valued and eliminated. It is possible for there to be valued and unvalued features that are interpretable or uninterpretable. However, when two features undergo an Agree relation, either one feature is interpretable (e.g., valued phi-features on a nominal) and the other is uninterpretable (e.g., unvalued phi-features on T), or both features are unintepretable (e.g., a valued uninterpretable case feature on T, 'u+Case:Nom', and an unvalued uninterpretable case feature on a nominal, 'uCase:_').

This model assumes that an EPP feature is a subfeature of a feature, along the lines of [10]. Specifically, a '+Feat:X' feature (feature b in Table 1) can have an EPP subfeature that forces an element with a matching unvalued feature to Merge directly with the element containing the '+Feat:X'; the EPP subfeature is only satisfied if the LI with a matching unvalued feature is Merged locally. For example, the English finite T contains a 'u+Case_EPP:Nom' feature. The EPP subfeature forces a subject, with an unvalued 'Case:_' feature, to undergo movement from the v* edge, assuming that it is base generated in v*; the subject leaves a copy in its base position and is re-Merged with T, thus eliminating the EPP subfeature on T and valuing the Case feature on the subject. The checked Case features are then eliminated because they are uninterpretable.

The Selector utilizes the following algorithm to choose an LI to select from the working subnumeration.

(5) *Select LI*:

(a) *If nothing is selected, select an N (or a predicate).*
(b) *Search for an LI that has an unvalued feature that matches the label of the tree. (Select the LI that has the fewest number of unvalued features first.)*
(c) *A subnumeration must be emptied.*

According to (5a), initially, an LI with the label N is selected.[1] Once an LI is selected, the selection process proceeds via application of (5b-c). The tree in the derivational working space has a label. Thus, in accord with (5b), there is a search in the current subnumeration for an LI with an unvalued label feature (e.g., 'N:_') that matches the label of the tree (e.g., 'N'). Note that the algorithim searches for LIs with the fewest number of unvalued features first, in accord with (5b). This is crucial to avoid indeterminancy in the selection process. For example, assume that the label of the tree is D and both V and v* are in the subnumeration. Both V and v* have 'uD:_' features, since V selects a D complement and v* selects a D subject. In this model, V has 2 unvalued features '[uTNS:_, uD:_]', where the unvalued 'uTNS:_ ' requires valuation from a matching valued feature on T. The

[1] In the absence of N, a predicate, such as a predicate adjective is selected.

v* has the 3 unvalued features '[uPhi:_, uD:_, uV:_]', where 'uPhi:_' represents unvalued uninterpretable phi-features and 'uV:_' is an uninterpretable unvalued V feature (v* selects for V). The v* has more unvalued features (3 versus 2 for V) because it selects for both a V and a D. Thus, V is Selected from the subnumeration. Lastly, (5c) requires that if no LI is selected from a subnumeration ((5a-b) fail), and there is only one LI remaining, that remaining LI is selected. In this manner, this model incorporates an explicit method for selecting LIs from a subnumeration.

After the selection process, there is a process of Merge that proceeds as follows.

(6) *Merge*:

(a) *If there is no tree for a selected LI to Merge with, then repeat the selection process.*
(b) *Otherwise, Merge the selected LI with the tree.*
(c) *The LI that initially has an unvalued label feature gives the tree its label.*
(d) *Otherwise, the label is the label of the tree.*
(e) *An unvalued label feature is valued by the label of the sister LI.*

At the initial stage of a derivation, once an element is selected, the selection process must be repeated to select one more element (6a), so that there will be two elements that can Merge. If there is already a tree in the derivational workspace, then the selected element is Merged with the tree. The label of a Merged syntactic object is determined according to (6c-d). Following (6c), if a tree with label Y Merges with an LI X, if X has an unvalued label feature 'Y:_', then the label becomes that of X, producing a structure of the form $[_X$ X Y] (a head-complement structure). In this case, the unvalued label feature of X is valued by the label of Y, in accord with (6e). If the label Y of a tree has an unvalued label feature 'X:_', then when Y Merges with X, the label remains Y, also in accord with (6c), resulting in $[_Y$ X Y] (a spec-head structure), and the unvalued label feature of Y is valued by the label of X. According to (6d), if the label Y of a tree and the selected LI X both lack any unvalued label features, then the label remains that of Y (an adjunction structure), resulting in $[_Y$ X Y].

After Merge, there is a feature checking operation (7).

(7) *Check Features*:

(a) *An unvalued 'Feat:_' feature is valued by a matching valued feature.*
(b) *If a probe (e.g., '+Feat_EPP:X') contains an EPP subfeature, feature checking requires local Merge.*

Feature checking results from a probe-goal relation, where a probe must be contained within the label LI of a tree. A probe is either a 'Feat:_' or a '+Feat:X' feature, which searches for an LI in the Merged structure that contains a matching feature. An LI with a 'Feat:_' searches for a matching valued feature, and an LI with a '+Feat:X' feature searches for a matching unvalued feature. If an LI with a matching feature is found within the accessible portion of a tree (a portion that has not been sent to Spell-Out), there is a feature valuation relation (7a), resulting in checked features. During the feature checking process, if a probe '+Feat:X' contains an EPP subfeature, then the element with the matching unvalued feature must be Merged locally (7b). If the externally Merged LI does not have a matching feature, then the EPP subfeature can force internal Merge. For example, a '+Case_EPP:Nom' on T forces a subject to leave a copy in its base position and be re-Merged to T.

This process of Selection, Merge, and feature checking is repeated until a subnumeration is emptied. Then, if the Merged structure does not have a phase label v* or C*, which is the case for adjuncts and subjects, it is reinserted into the higher subnumeration (8), after which it functions as any other LI and can be selected and re-Merged.

(8) *Reinsert a complete non-phase into a higher subnumeration.*

Otherwise, these processes of Selection, Merge, and feature checking continue until Spell-Out, which applies in the following two conditions: a) Spell-Out applies to a lower phase when the head of a higher phase is Merged (in (9), when C* is Merged, the lower v* phrase is sent to Spell-Out), and b) Spell-Out applies when a numeration is emptied. I use * to indicate a phase head - thus v* and C* are phase heads.

(9) $[_{C*}$ C* $[_T$ T $[_{v*}$ v* $[_{v*}$ V...]]

Note that an entire phase, not just the complement of a phase head, is sent to Spell-Out at once (the entire v* phase in (9) is sent to Spell-Out when C* is Merged). Thus, there is a one-to-one correspondence between phase-hood and Spell-Out.

Once a phrase is sent to Spell-Out, there is first a Last Resort Check[2] operation (10) whereby an element that is contained within a phrase that is about to be sent to Spell-Out is reinserted into a higher subnumeration as a Last Resort, thus saving a derivation from crashing.

(10) *Last Resort Check: If an LI with an unvalued feature is contained within a phrase that is about to be sent to Spell-Out, reinsert this LI into the current working subnumeration.*

In (11), when C* is Merged, the lower v* will be sent to Spell-Out. If α contains an unvalued feature, then when C* is Merged, α is reinserted into the higher subnumeration headed by C*. The renumerated LI α then functions as any other LI - it can be selected and re-Merged into a derivation.

(11) $[_{C*}$ C* T $[_{v*}$ v* V $\alpha_{[Feat:\_]}$]]

This operation crucially does away with the need to posit EPP features that exist solely for the purpose of bringing an element to a Phase Edge (see section I).

Once the Last Resort Check operation is completed, there is a Crash Check operation (12).

[2]I thank Sandiway Fong, with whom I developed this view of Last Resort. Any problems with this view are my own.

(12) *Crash Check*: *If there are any unvalued features 'Feat:_' or any unassigned valued features '+Feat:X', the derivation will crash.*

If the derivation does not crash, then the relevant phrase (an unordered set that contains LIs and other sets of LIs) is linearized via recursive application of the algorithm in (13).

(13) *Linearization*:

(a) *Find the most embedded phrase.*

(b) *A label LI is Merged to the left when the label LI Merges for the first time.*

(c) *A non-label LI Merges to the left when the label LI has previously undergone Merge.*

(d) *Reinsert the linearized element into the higher phrase.*

The Selector chooses the most embedded phrase (a set of Merged LIs), linearizes it and then reinserts it into the larger phrase that it is embedded in (if present). For English, when a label LI Merges with another LI, and nothing has Merged with the label before (first Merge), then, in accord with (13b), the label is Merged to the left (a head precedes its complement). Following (13c), Merge of another element to the label (second Merge) is also to the left (a specifier or adjunct precedes a head).[3] Once a phrase is linearized, phonological rules apply; for example, T may be pronounced as a form of 'do', etc. Since a derivation is built in a bottom up fashion, a successfully linearized phrase that is contained within a larger phrase is not actually pronounced until the derivation is complete.

Lastly, this model incorporates a model of *wh*-phrases that follows work by Cable [11]. Cable, based primarily on evidence from Tlingit, proposes that in a *wh*-question, a question particle 'Qu'[4] Merges with a *wh*-phrase, and furthermore, C attracts Qu rather than a *wh*-phrase. In a *wh*-movement language (such as English), a *wh*-phrase is the complement of Qu, and thus when Qu moves, it brings its complement *wh*-phrase with it. In this computational model, an English *wh*-phrase is is the complement of Qu. I demonstrate how this works in the following section.

III. Derivation

In this section, I demonstrate how this model successfully produces the derivation of the 'simple' English *wh*-question (14). The computer model is fed the underlying numeration (15), where 'C_Int' refers to an interrogative C and 'PAST' is a past tense T.[5]

(14) What did John eat?

(15) {C_Int, PAST, {v*,{D, John}, Qu, D, what, eat}}

Crucially, the numeration and embedded subnumerations of (15) are sets. The computer program automatically constructs a detailed step-by-step derivation of (15).

The model utilizes a 'lexicon' to automatically render each LI into a list consisting of the label, the form, and the features that are relevant for the derivation. For example, the LI 'John' has the representation '[N, John, [iPhi:X]]' in which its label is 'N', its form is 'John', and it has interpretable and valued phi-features 'iPhi:X'.

Initially, the Selector finds the most embedded subnumeration, in accord with (4), and thus finds the subnumeration containing the subject, which requires its own subnumeration (see section I). In (16), the LIs in the subnumeration are represented as lists containing the label, form, and features.[6]

(16) {[D, [uN:_, uCase:_]], [N, John, [iPhi:X]]}

Since no LIs have been selected (as this is the initial stage of the derivation), the N 'John' is selected, in accord with (5a). As only one element has been selected at this point, the selection process is repeated, following (6a). The D is then selected in accord with (5b) because it contains an unvalued label feature 'uN:_' that matches the label of the already selected N. The two LIs Merge and, following (6c), since D initially contains an unvalued label feature 'uN:_', the label of the newly formed syntactic object is D. At this point, the unvalued label feature 'uN:_' of D is valued by the N label of 'John', in accord with (6e). Since this label feature is uninterpretable, it is eliminated. The resulting Merged structure (a set that lacks linear order) is shown in (17). The Merged LIs are enclosed in brackets containing the label, form, and features, and the head of the Merged structure is the non-bracketed D. The 'Chkd' (for 'checked') in 'ChkdN:X ' refers to an uninterpretable feature that has been deleted.

(17) {D, [D, [ChkdN:X, uCase:_]], [N, John, [iPhi:X]]}

Since the subnumeration is emptied and this syntactic object with the label D is a non-phase, following (8), there is a process of renumeration - the D phrase is reinserted into the higher v* subnumeration.

The Selector then moves on to the next higher v* subnumeration (18), which contains the renumerated subject (a set with the label D).

(18) {[V, eat, [uTNS:_, uD:_]], [N, what, [iPhi:X, iWH:X]], [D, Qu, [uTyp:Int, iScp:_, uCase:_, uN:_]], [v*, [u+Case:Acc, uPhi:_, uD:_, uV:_]], {D, [D, [ChkdN:X, uCase:_]], [N, John, [iPhi:X]]}}

At this point, the N 'what' is selected, following (5a).[7] Since there is only one selected element, the selection process is repeated (6a) and a Q-particle 'Qu' is selected, in accord with (5b), since it contains an unvalued label feature 'uN:_'. I assume, for the sake of simplicity, that Qu is a D.[8] The process of Merge and feature checking apply, thereby resulting in the Merged structure in (19), with the label D, due to D's initial unvalued label feature 'uN:_ ' (6c).

[3]The linearization order may require some variation with respect to certain adjuncts. I leave this issue for futher research.

[4]Cable used 'Q'.

[5]I assume that in English if there is an N, there there must be a D.

[6]I assume that an unvalued case feature 'uCase:_' occurs on D rather than N. This issue though requires further examination.

[7]The 'iWH:X' feature represents a valued interpretable *wh*-feature.

[8]One possibility is that a D head Merges wth Qu.

(19) {D, [D, Qu, [uTyp:Int, iScp:_, uCase:_, ChkdN:X]], [N, what, [iPhi:X, iWH:X]]}
Qu contains a 'uTyp:Int', which is an uninterpretable clausal typing feature 'uTyp' with the value 'Int' for interrogative. Qu also contains an unvalued 'iScp:_' feature, that when valued, gives a *wh*-phrase scope. In this model, the interrogative C* contains an unvalued 'iTyp:_' feature and a valued 'uScp:X' feature. When C and Qu undergo a feature valuation relation (at the final stage of this derivation), the 'uTyp:Int' of Qu values the 'iTyp:_' of C, thereby giving the clause an interrogative interpretation, and the valued 'uScp:X' of C values the 'iScp:_' of Qu, giving the *wh*-phrase scope.[9]

The remainder of the v* phrase is constructed as follows. There is again a search within the subnumeration for an element that 'selects' a D, which is the label of the tree. Although both V and v* have 'uD:_' features, V is selected in accord with (5b), since it has fewer unvalued features than v*. After Merge, the label of the tree is V, since V initially contains an unvalued label feature (6c). Another search within the subnumeration results in selection of v*, since it contains a 'uV:_' feature (5b) that matches the V label of the tree. After Merge, the label is v* since v* contains an unvalued label feature (6c). Feature checking applies and the unvalued phi-features 'uPhi:_' on v* function as a probe that Agrees with and is valued by the valued phi-features 'iPhi:X' on the N 'what'. The 'u+Case:Acc' on v* also acts as a probe that Agrees with and values the unvalued 'uCase:_' feature on D. Since case is uninterpretable on both v* and D, the accusative case features are eliminated from the derivation. At this point, the subnumeration only contains one element - the renumerated subject. In accord with the requirement that a subnumeration be emptied (5c), the renumerated subject is Selected and Merged. Since the tree label v* initially contains an unvalued label feature 'uD' - the label remains v* (6c).

The v* subnumeration has been emptied, and so the Selector works on the next higher subnumeration (20).
(20) {[C*, [iTyp:_, uT:_, i+Force_EPP:X, u+Scp_EPP:X]], [T, PAST, [u+Case_EPP:Nom, uPhi:_, i+TNS:PST, uv*:_, uForce:_]]}
The Selector chooses T in accord with (5b), since T contains an unvalued 'uv*:_' feature. After Merge, the label is T due to T's unvalued label feature (6c). Next, there is a feature valuation relation (7a) in which T's unvalued 'uPhi:_' feature functions as a probe and Agrees with, and is valued by, the valued 'iPhi:X' of the subject 'John'. T also contains the feature 'u+Case_EPP:Nom', which is a '+Feat:X' feature with an EPP subfeature. The EPP subfeature prevents an Agree relation alone from allowing this feature to value a matching unvalued Case feature. In accord with (7b), the EPP subfeature forces the subject, which contains a matching unvalued 'Case:_' feature, to leave a copy in its base position and re-Merge with T. The label remains T, in accord with (6d), since neither T nor the subject D contains an unvalued label feature. Next, the phase head C* is selected from the subnumeration, since it contains a 'uT:_' feature (5b).

When C* is Merged, the lower v* phase must be sent to Spell-Out, assuming that when a higher phase head is Merged a lower phase is sent to Spell-Out (see section II). Initially, there is a Last Resort Check operation (10); there is a search for an element in the lower v* phrase that has an unvalued feature. Since Qu contains an unvalued 'iScp:_' feature, Qu (along with its complement *wh*-phrase) is reinserted into the current subnumeration and a copy is left in the base position. Next, there is a Crash Check operation (12) whereby there is a search for unvalued or unassigned features. In this case, none are found (copies are not visible to this operation) and the v* phrase is linearized. Each set contained within v*, beginning with the most embedded set, is linearized according to the algorithm in (13). First Merge between a label LI and another LI results in the label occuring on the left, and any further Mergers (second Merge) to the label result in the label occuring on the right. Phonological rules apply and the tree in Figure 1 is produced, which is automatically drawn by the computer program. Note that this tree contains linearized copies of both the subject 'John' and *wh*-phrase 'what'. The line through the v* edge signifies that the v* phrase has been sent to Spell-Out. Uninterpretable features that have been eliminated are preceded by the prefix 'Chkd'.

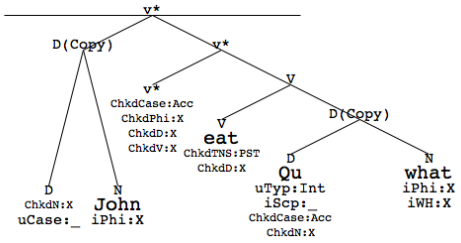

Figure 1. Linearized v* Phrase

The linearization process for the v* phrase is complete and the derivation continues. The interrogative C* (see (20)) contains two '+Feat:X' features that each contain an EPP subfeature; a 'i+Force_EPP:X' feature and a 'u+Scp_EPP:X' feature. The '+Force:X' feature, a modified version of Force proposed by [15], is associated with determining whether or not a clause is matrix or embedded by matching an unvalued feature in T, and the '+Scp' feature of C* is associated with giving an element scope by valuing a matching Scope feature, in this case on Qu. Initially, the 'i+Force_EPP:X' feature agrees with an unvalued matching 'uForce_' feature on T and the EPP subfeature forces the matching LI, in this

[9]The notion that Qu is responsible for clausal typing is discussed in [12], [13], among others. I posit the existence of a scope feature in C based on the fact that CP is a projection where clauses obtain scope [14].

case T, to leave a copy in its base position and to re-Merge with C*, following (7b), thus successfully modeling T to C movement in English matrix interrogatives. This re-Merge operation results in feature-checking of the Force feature and elimination of the EPP subfeature.[10] Remember that the subnumeration now contains the Qu/*wh*-phrase 'what', which was renumerated as a Last Resort. This LI is the only element left in the subnumeration, and thus it is selected, in accord with (5c). The label remains the same, C*, since neither of the Merged elements has an unvalued label feature (6d). After selection and Merge, feature checking results in the 'iScp:_' feature of Qu being valued by the 'uScp:X' of C*, thereby giving the Qu/*wh*-element scope. In addition, the 'iTyp:_' feature on C is valued by the 'uTyp:Int' on Qu, thereby typing the clause as an interrogative.

The entire numeration is emptied and the Spell-Out operations (10-13) apply, followed by application of phonological rules, which result in T on C being pronounced as 'did', as shown in the final tree in Figure 2.

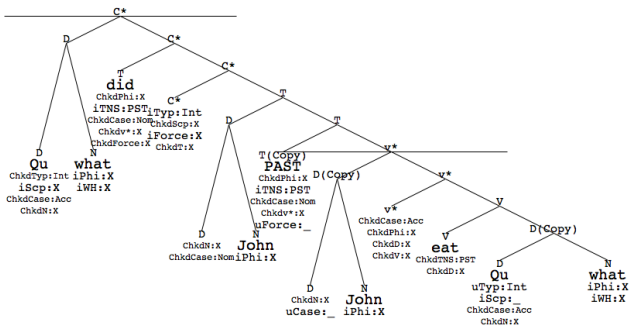

Figure 2. Linearized C* Phrase

IV. Conclusion

I have demonstrated how this computational model succesfully derives the syntactic representation of the English *wh*-question (14). I leave for future work in depth discussion of the many details of this model, as well as its application to a variety of other examples in English and other languages. Crucially, there are several ways in which this model helps to clarify our understanding of how language is generated, from the perspective of Phase Theory. This model a) provides a specific algorithm for selecting elements from a numeration/subnumeration, b) provides a specific algorithm for determining the label of a Merged syntactic object, c) incorporates a one-to-one correspondence between phase-hood and the timing of Spell-Out, and d) eliminates the need for superfluous EPP features - via a Last Resort process. This model thus make progress towards clarifying how Phase Theory works, and makes progress towards discovering precisely how the human mind generates sentences.

[10]I assume that the interrogative C* contains a '[+Scp_EPP:X]' with an EPP subfeature in order to account for local movement of subject *wh*-phrases, which due to their close proximity to C*, cannot result from the Last Resort process.

Acknowledgment

I would like to thank Sandiway Fong, Simin Karimi, and the three anonymous reviewers. All errors are my own.

References

[1] N. Chomsky, *Derivation by phase*. MIT Working Papers in Linguistics: MIT Occasional Papers in Linguistics Number 18, 1999.

[2] ——, "Minimalist inquiries: The framework," in *Step by step*, R. Martin, D. Michaels, and J. Uriagereka, Eds. Cambridge, MA: MIT Press, 2000, pp. 89–155.

[3] ——, "Beyond explanatory adequacy," in *Structures and beyond: The cartography of syntactic structures, volume 3*, A. Belletti, Ed. Oxford, UK: Oxford University Press, 2004, pp. 104–131.

[4] ——, "On phases," in *Foundational issues in linguistic theory; essays in honor of Jean-Roger Vergnaud*, R. Freidin, C. Otero, and M.-L. Zubizaretta, Eds. Cambridge, MA: MIT Press, 2006, pp. 133–166.

[5] G. Grewendorf and J. Kremers, "Phases and cyclicity: Some problems with phase theory," *The Linguistic Review*, vol. 26, pp. 385–430, 2009.

[6] K. Johnson, "Towards an etiology of adjunct islands," *Ms., University of Massachusetts at Amherst*, 2002.

[7] C. Felser, "Wh-copying, phases, and successive cyclicity," *Lingua*, vol. 114, pp. 543–574, 2004.

[8] N. Chomsky, "Bare phrase structure," in *Evolution and revolution in linguistic theory*, H. Campos and P. Kempchinsky, Eds. Washington, D.C.: Georgetown University Press, 1995, pp. 51–109.

[9] ——, *The Minimalist Program*. Cambridge, MA: MIT Press, 1995.

[10] D. Pesetsky and E. Torrego, "T-to-C movement: Causes and consequences," in *Ken Hale: A life in language*, M. Kenstowicz, Ed. Cambridge, MA: MIT Press, 2001, pp. 355–426.

[11] S. Cable, "The grammar of Q: Q-particles and the nature of Wh-fronting, as revealed by the wh-questions of Tlingit," Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[12] J. Katz and P. Postal, *An integrated theory of linguistic descriptions*. Cambridge, MA: MIT Press, 1964.

[13] J. Aoun and Y. Li, "Wh-elements in situ: Syntax or LF," *Linguistic Inquiry*, vol. 24, pp. 199–238, 1993.

[14] R. May, *Logical Form*. Cambridge, MA: MIT Press, 1985.

[15] L. Rizzi, "The fine structure of the left periphery," in *Elements of grammar*, L. Haegeman, Ed. Dordrecht: Kluwer, 1997, pp. 281–337.