社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

A Phase Theory-Based Computational Model of Sentence Generation

Jason GINSBURG†

†Center for Language Research, University of Aizu Aizuwakamatsu, Fukushima, 965-8580 Japan

E-mail: †jginsbur@gmail.com

Abstract This paper presents a computational model of sentence generation, based on Phase Theory, that attempts to model on a computer the innate structures of the human mind that are utilized for language generation and processing. This model automatically constructs the derivation of a sentence from a numeration, which is a set that consists of subnumerations (language chunks) that contain lexical items. This model presents a simple algorithm for selection and Merge of lexical items from subnumerations to create sentences of English and Japanese.

Keywords Syntax, Phase Theory, Computational Modeling

1. Introduction

This paper presents a computational model of sentence generation, based on recent work in Phase Theory ([1-3]) that attempts to replicate on a computer the innate structures of the mind that enable humans to produce and understand sentences. We created a computer model (implemented in Python) that produces detailed step-by-step derivations of sentences from underlying numerations. Our goal is to create a more refined and precise theory of language generation, based on Phase Theory, that accounts for how sentences are generated in various languages. In this paper, we discuss how this computational model constructs a derivation from a numeration and we demonstrate how this model constructs the derivations of some English and Japanese sentences.

2. Assumptions

In Phase Theory, a sentence is constructed from the bottom-up via a process of selection and Merge of Lexical Items (LIs) from a numeration. A numeration consists of subnumerations, and a subnumeration is a set of LIs that can corresponds to a phase, with a phase head v* or C*,[1] or to a subject or adjunct, which must be constructed in a separate derivational space from the main spine of a derivation (cf. [4]), since a subject/adjunct can be a complete clause, as in "**That she won an award** bothered him."

A derivation is subject to the Phase Impenetrability Condition (PIC), which has been formulated in at least two ways that differ with respect to the timing of Spell-Out. In one version [1-2] in (1), when the phase head *v** is Merged, the complement of v*, the VP, is sent to Spell-Out. In the other version [3], when the phase head C* is Merged, the complement of *v**, the VP, is sent to Spell-Out.[2]

(1) [CP C* [TP T [*v*P* *v** [VP V ...]]]]

Once a phrase is sent to Spell-Out, its contents are no longer accessible to higher operations.

Another important component of Phase Theory is the idea that a head can function as a probe that Agrees with a matching goal, where the probe c-commands (or at least is higher in the tree than) the goal, thus resulting in a feature checking relation. Crucially, a probe can only Agree with a goal that has not been sent to Spell-Out.

Our computational implementation of sentence generation utilizes a modified version of Phase Theory that incorporates: a) an explicit theory of feature valuation, b) explicit methods for selection of LIs from a subnumeration, and c) a one-to-one correspondence between 'phase-hood' and Spell-Out.

Feature valuation is an important component for driving a derivation. Our model incorporates the types of features in (2). There are unvalued (2a) and valued (2b-c) features. Unvalued features (2a), where the ":_" signifies the lack of a value, must be valued by forming an Agree relation with matching valued features (2b-c), where the ":X" signifies a value. An example of an

1 The * indicates a phase head; thus v* and C* are phase heads.

2 See [5] for discussion of the different versions of the PIC.

unvalued feature of type (2a) would be unvalued phi-features

(2) Features:

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

'[Phi:_]' on T that must be valued by the matching valued '[Phi:X]', of type (2c), of a subject. Valued features come in two varieties. One variety (2b), contains a "+" which signifies that this feature must value a matching unvalued feature. We model theta-roles and Case features as features of this type. For example, "*I like" is ill-formed because the verb 'like' has '[+Theta:X]' and '[+Case:ACC]' features that are unable to value matching unvalued features, since there is no object. A valued feature of type (2c) also can value a matching unvalued feature, but there is no inherent need for it to do so – thus it lacks a '+'. Phi-features '[Phi:X]' on a nominal are of this type. Most features that determine the meaning of an LI are probably of type (2c). For example, an N may have various features that determine its meaning (e.g., features that determine that the LI 'dog' refers to a four legged mammal), but that do not need to undergo any feature valuation relations. An unvalued feature (2a) or an unassigned valued feature (2b) can function as a probe, when it occurs in the label LI (head) of a tree, that searches for a matching goal within a Merged tree structure. Crucially, a derivation will crash if any unvalued features (2a) or unassigned valued features (2b) are sent to Spell-Out.[3]

In the Minimalist Program [6], as well as in Phase Theory, movement of LIs is often accounted for with the EPP feature, which is thought to motivate movement of subjects, as well as other elements; [1] suggests that EPP features in phase heads force movement of a wh-phrase in languages such as English.

3 In the Minimalist Program [6], as well as in Phase Theory, it is generally assumed that there are interpretable and uninterpretable features that appear on LIs. Interpretable features must remain to the level at which a derivation is interpreted (LF), whereas uninterpretable features must be eliminated from a derivation before the level of interpretation. We make no claims about whether or not features are eliminated. Rather, there must be no unvalued features and no unassigned features at the point at which a syntactic structure is interpreted.

We incorporate EPP features into our model as subfeatures of features, along the lines of [7]. In our approach, a feature of type (2b), a '[+Feat:X]' feature (a feature that must be assigned), can have an EPP subfeature which requires that an LI with a matching unvalued feature Merge locally. For example, the English finite T has a '[+Case_EPP:NOM]' feature, a valued nominative feature with an EPP subfeature. In a successful derivation, the '[+Case_EPP:NOM]' feature of T probes for and finds a matching unvalued feature in a subject at the v*P edge. The EPP subfeature forces the subject to undergo movement and re-Merge with T. The result is that the Case feature of T values the unvalued Case feature of the subject, and the EPP subfeature is eliminated.

In Phase Theory, it is assumed that elements are selected from subnumerations. However, it is not clear why they are selected in the particular order that they are selected in. Assuming that (3a), with the numeration in (3b), is built from the bottom-up, 'dinner' is selected before the verb 'ate'. However, the subnumeration of the v*P, contained in the numeration (3b), is an unordered set of LIs (where the subject has its own subnumeration), which raises the issue of how the grammar module 'knows' what to select first. Why would 'dinner' be selected before v*, etc.?

(3) (a) I ate dinner. (b) {T, C*,{{I}, v*, ate, dinner}}}

In our model we propose an explicit algorithm, given in (4), for selection of LIs from a subnumeration. Crucially, our algorithm sucessfully derives bottom-up selection of LIs from a subnumeration.

(4) (a) If the label LI of a tree contains an unvalued feature (2a), or a valued feature that must be assigned (2b), then search for an LI in a subnumeration with a matching feature.
(b) Otherwise, search for an LI in a subnumeration according to a default selection order: N > D > V > v/v* > T > C*.

In accord with (4a), if the label LI of a tree is v* with a '[+Theta:X]' feature, there is a search within a subnumeration for an LI with a matching '[Theta:_]' feature. If an LI, such as a subject, with a '[Theta:_]' feature is found, then it is selected and Merged with the tree. When (4a) cannot apply, either because there is nothing in the derivational workspace (at the initial stages of a derivation) or because the head of a tree lacks unvalued features

(2a) or valued features that must be assigned (2b), then the language component relies on a default selection order (4b). There is a search within a subnumeration for LIs with particular labels in the order given in (4b), where labels on the left are searched for before those on the right. For example, first there is a search for an LI with label N. If no LI of this type is found, then there is a search for a D, and so on. The default selection order (4b) is a hypothesis about the way that language is structured in the brain. There is a default selection order that the human mind relies on in an elsewhere condition, when (4a) cannot apply.

Notable in Phase Theory is that there is a lack of correspondence between phases and Spell-Out. According to the PIC (see section 1), the complement of a phase head is sent to Spell-Out separately from its associated phase head. Thus, there is not a one-to-one correspondence between a subnumeration of a phase and Spell-Out. Our model takes a different approach. When a phase head is Merged, a lower phase, if present, is sent to Spell-Out. For example, in (5), when the phase head C* is Merged, the entire lower v* phase is sent to Spell-Out.

(5) [C* [T [*v** V]]]

In this manner, all elements of a phrase that originate in a single subnumeration are sent to Spell-Out as a single unit.[4]

Our model thus incorporates a modified version of Phase Theory that provides explicit algorithms for feature checking, selection of LIs, and the timing of Spell-Out.

3. Implementation

We created a computer program, in the Python programming language, that automatically builds a derivation of a sentence from a numeration. In this section, we briefly discuss the algorithms that our computational implementation utilizes.

Our model utilizes a bare phrase structure [9] in which a tree is constructed via iterative Merge of LIs. After each Merger, the newly formed tree has the label of one of the Merged elements. First Merge with an LI results in a head-complement structure, and further Merge operations with the same LI result in head-spec/adjunct structures.

A derivation is constructed using the following algorithms.

(6) **Select a Subnumeration**: Select the most embedded subnumeration with a phase head. Search for any further embedded subnumerations (e.g., a subject/adjunct).

(7) **Select an LI:** If the label LI of a tree has an unvalued feature (2a) or a valued feature that needs to be assigned (2b), search for an LI in the subnumeration with a matching feature (see 4a). Otherwise search for an LI according to the default selection order (4b). Repeat the selection process if there is only one element selected (Merge requires two LIs).

(8) **Merge**: Merge the selected LI with the tree (the element in the derivational working space). Whichever label (of an LI) has a "+" feature is the label of the tree. If neither LI has a "+" feature, the label remains the same.

(9) **Feature Checking**: The label LI of a tree undergoes feature valuation relations, if possible, with matching features in a tree. An unvalued feature can only be valued once.

(10) **Repeat**: Repeat the process of selection and Merge until a subnumeration is emptied.

(11) **Reinsertion**: If a subnumeration is not a phase (i.e., a subject or adjunct), then reinsert into a higher subnumeration.

(12) **Spell-Out**: If a phase head is Merged, send a lower phase (if present) to Spell-Out.

(13) **Crash Check**: Crash (end the derivation) if there are any unvalued features (2a) or unassigned features (2b) in the Spell-Out domain.

(14) **Linearize**: If the derivation does not crash, linearize the Spell-Out domain. Convert the Merged phrase into a syntactic object with linear order and apply phonological rules.

(15) **Repeat**: Continue repeating these processes (6-14), until the entire numeration is emptied and the complete sentence is

4 The PIC crucially makes the edge of a phase accessible to a higher phase, thus allowing movement (e.g., of a wh-phrase) out of a phase. A problem with this view is that something must force an element such as a wh-phrase to move to the edge of a phase. [1] suggests that a phase head can optionally have an EPP feature that serves this purpose. Our model does not utilize EPP features that exist simply for the purpose of bringing an LI to a phase edge. Although we do not have space to discuss the details here, we account for certain movement phenomena out of phases as follows. An LI (such as a wh-phrase) with an unvalued feature, when contained within a phase that is about to be sent to Spell-Out, can be reinserted into a subnumeration as a last resort, and then selected and re-Merged in a higher position. See [8] for details of this approach.

linearized, or until the derivation crashes.

Our model utilizes the algorithms in (6-15) to select and Merge elements in order to create the derivations of sentences. We next demonstrate how this model automatically constructs 'simple' English and Japanese statements.

4. English

Our model successfully constructs the derivation of the statement in (16a) when fed the numeration in (16b), where each subnumeration is enclosed in set brackets. The numeration (16b) is a set (of unordered elements) with the phase head C*. The main numeration contains an embedded subnumeration with the phase head v*. The v* subnumeration contains a subject, which has its own subnumeration (see section 1).[5]

(16) (a) I eat food.

(b) {C*, PRES,{v*,{I, D}, eat, D, food}}

The computer program uses a "lexicon" to render (16b) into the form in (17), where each LI is a list consisting of the label, form, and (relevant) features of that LI. For example, the LI 'food' has the label 'N', the form 'food', and the features '[Theta:_ Phi:X Case:_ DEF:_]' ('food' has unvalued theta, case, and definiteness (DEF) features, and valued phi-features).

(17) {{[T, PRES, [+Case_EPP:Nom Phi:_ Force:_]], [C* [+Force:X]]},{{[D, [D [+DEF:X]], [N I [Theta:_ Phi:X Case:_ DEF:_]]}}, [v* [+Theta:X +Case:Acc Phi:_] [V eat [+Theta:X]]},[N food [Theta:_ Phi:X Case:_ DEF:_]], [D [+DEF:X]]}

The computer program applies the algorithms in (6-15) to automatically construct a derivation from (17).

The derivation begins with selection of a subnumeration by the Selector, the component of the human mind that selects elements from a numeration. The Selector, in accord with (6), selects the most embedded subnumeration with a phase-head, in this case, the v* subnumeration, and then the Selector searches for a further embedded subnumeration. It finds the subnumeration of the subject 'I'. The Selector then chooses LIs in accord with (7). Initially, the Selector chooses the N, following the default selection order (4b). Then it chooses the D, again in accord with the default selection order. D and N Merge, and in accord with (8), the label of the Merged syntactic object is D, since D has a "+" feature of type (2b). There is a feature checking relation (9), whereby the '[+DEF:X]' feature of D values the matching unvalued '[DEF:_]' feature of N - D assigns a definiteness value to N. The subject, not being a phase, is reinserted into the v* phase (11). The Selector then works on the v* subnumeration. The processes of selection, Merge, and feature checking continues (10), resulting in formation of a complete v* phrase. Note that v* undergoes Merge twice. When v* is initially Merged, it is selected due to the default selection order (4b). After Merge with V, and initial feature checking (9) that results in Case assignment to the object 'food', v* still contains a '[+Theta:X]' feature. Thus, in accord with (7), there is a search in the subnumeration for an LI with a matching unvalued feature, and the renumerated subject, with its matching unvalued '[Theta:_]' feature, is selected and Merged.

Since the v* subnumeration is emptied, the Selector works on the higher subnumeration containing T and C*. T is selected in accord with the default selection order (4b). T contains '[+Case_EPP:Nom]' and '[Phi:_]' features. After Merge of T, there is a feature checking operation (9) in which the '[Phi:_]' feature of T probes with and Agrees with the valued '[Phi:X]' feature of the subject 'I'. The [+Case_EPP:Nom]' feature finds and Agrees with the unvalued '[Case:_]' of the subject and the EPP subfeature forces the subject to undergo movement. The subject leaves a copy in its base position and it re-Merges with T, resulting in elimination of the EPP on T and valuation of the Case feature on the subject 'I'.

Next, following the default selection order (4b), C* is Selected and the resulting Merged structure is (18), a Merged syntactic object (automatically created by the computer model) that contains unordered sets of Merged LIs.

(18) {C*, [C* [+Force:X]], {T, {T, [T PRES [cCase:Nom cPhi:X Force:_]], {v*, {v*, [v* [cTheta:X cCase:Acc cPhi:X]], {V, [V eat [cTheta:X]], {D, [D [cDEF:X]], [N food [cTheta:X cPhi:X cCase:Acc cDEF:X]]}}}, {D(Copy), [D [cDEF:X]], N I [cTheta:X Phi:X Case:_ cDEF:X]]}}}, {D, [D [cDEF:X]], [N I [cTheta:X cPhi:X cCase:Nom cDEF:X]]}}}

Checked features are indicated with the "c" prefix. For example,

5 We assume that an N must occur with a D, even if the D is silent.

'[cTheta:X]' on v* and on the subject 'I' indicates that both of these elements have valued theta-features.

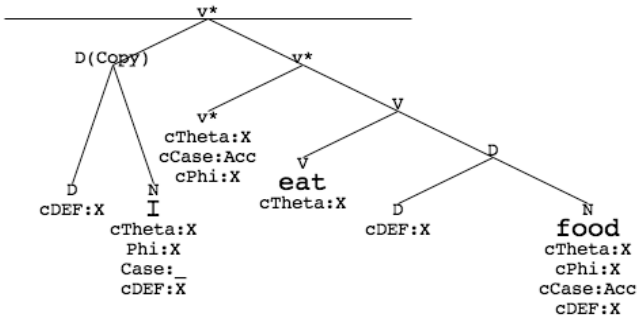
Once C* is Merged, before any feature checking relations can apply, the embedded v* phrase is sent to Spell-Out, in accord with (12), assuming that when a phase head is Merged, a lower phase is sent to Spell-Out. The crash check operation (13) finds no unvalued or unassigned features (unvalued features in the copy of the subject do not count), and the phrase is linearized (14). The linearization algorithm for English is given in (19).

(19) (a) First Merge: A label LI is Merged to the left when the label LI Merges for the first time.

(b) Second Merge: A non-label LI is Merged to the left when the label LI has previously undergone Merge.

First Merge (19a) results in the label LI appearing to the left (a head complement structure) and second Merge (19b) results in the non-label appearing to the left (a spec-head structure). The v* phrase that results after linearization is shown in (20), a tree that is automatically drawn by our program. The line through the v* edge signifies that the v* phrase has been sent to Spell-Out and linearized. Also note this phrase contains a linearized copy of the subject – the actual subject has moved to the T edge.

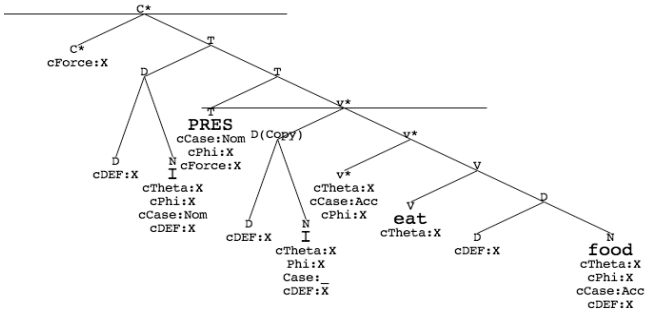
(20)



Once the v* phrase is completed, feature checking (9) occurs in the matrix clause. C contains a [+Force:X] feature (see 18) that values a matching unvalued '[Force:_]' feature on T - we assume that Force determines whether a clause is matrix or embedded.[6] Then, since the entire numeration has been emptied, the Spell-Out process applies again. The derivation does not crash (13) and linearization (14) applies, resulting in the well-formed tree in (21), where the line through the C* edge indicates that the tree has been linearized.

6 This is a modified version of Force proposed by [10].

(21)



5. Japanese

We next turn to the derivation of the simple Japanese statement (22a), which has the numeration in (22b).[7]

(22) (a) Kare-ga tabemono-o tabeta.

He-NOM food-ACC ate

'He ate food.'

(b) {C*, PAST, {v*, taberu, {kare}, tabemono}}

The numeration (22b) is rendered into the more detailed format in (23), in which each LI is a list consisting of the label, form, and features.

(23) {[T PAST [+Case_EPP:Nom Phi:_ Force:_]], [C* [+Force:X]], {[V, taberu, [+Theta:X]], {[N kare [Theta:_ Phi:X Case:_]]}, [N tabemono [Theta:_ Phi:X Case:_]], [v* [+Theta:X +Case:Acc Phi:_]]}}

The derivation, beginning with (23) is carried out in a virtually identical manner to that of the previous English example (16). The Selector finds the embedded v* subnumeration, in accord with (6). The v* phrase is constructed via the continuous process of selection (7) and Merge (8) of LIs, feature checking (9), and reinsertion of the subject (11). This is followed by selection and Merge of T. As in English, the EPP subfeature of '[+Case_EPP:Nom]' on T forces the subject to leave a copy in the v* phrase and re-Merge with T.[8] Once C* is Merged, the resulting structure is (24).

(24) {C*, [C* [+Force:X]], {T, {T, [T PAST [cCase:Nom cPhi:X Force:_]], {v*, {v*, [v* [cTheta:X cCase:Acc cPhi:X]], {V, V taberu [cTheta:X]], [N tabemono [cTheta:X cPhi:X cCase:Acc]]}}, {N(Copy), kare, [cTheta:X Phi:X Case:_]}}},

7 We assume that in Japanese, an N can occur without a corresponding D.

8 We assume that there is an EPP feature in T in Japanese, following [11].

{N, kare, [cTheta:X cPhi:X cCase:Nom]}}}

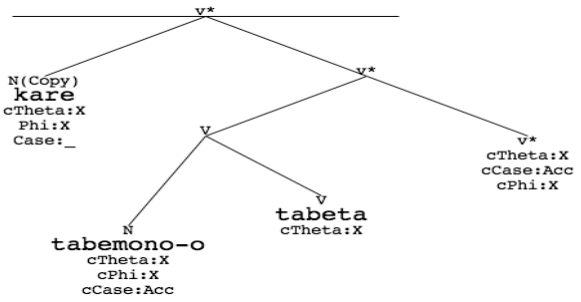
Since the phase head C* is Merged, the lower v* phrase is sent to Spell-Out (12). The crash check operation (13) finds no unvalued or unassigned features (the copy of the subject is ignored), and the linearization process (14) applies. The linearization algorithm for Japanese is as follows.

(25) (a) First Merge: A label LI is Merged to the right when the label LI Merges for the first time.

(b) Second Merge: A non-label LI is Merged to the left when the label LI has previously undergone Merge.

(25) results in heads being Merged to the right of complements and specifiers being Merged to the left.[9] The resulting linearized v*P, automatically constructed by our program, is shown in (26). Note that application of phonological rules results in the verb *taberu* 'eat' being pronounced in the past tense form *tabeta* 'ate', due to the occurrence of the Past tense T in the higher phrase, and the accusative case particle *-o* appears on the object *tabemono* 'food'.

(26)

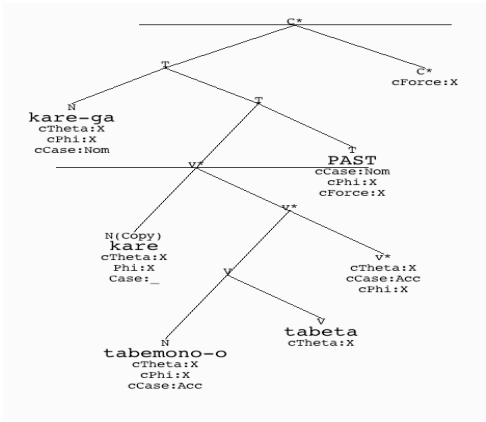


Next, C* and T undergo a feature checking relation (9), resulting in valuation of the Force feature on T. Since the numeration is empty and all feature checking operations have completed, the entire phrase is sent to Spell-Out, where it is linearized and phonological rules apply (the nominative case particle *-ga* appears on the subject), resulting in (27).

5. Conclusion

We have demonstrated how this model derives the structures of two simple sentences in English and Japanese using precise algorithms, based on Phase Theory, for selecting and Merging LIs from a numeration and for sending phrases to Spell-Out at specific times in the course of a derivation. This model shows that a limited amount of built in grammatical structure can be utilized to create sentences in different languages. In future work, we hope to extend this model to generate a wide variety of sentences in English, Japanese, and other languages. We hope that this model can be utilized to help further understanding of how humans produce and process language.

(27)



9 This ordering algorithm is suitable for the relevant Japanese example, but it may require some refinement for other constructions.

REFERENCES

[1] N. Chomsky, Derivation by Phase, MIT Occasional Papers in Linguistics Number 18, Cambridge, MA, 1999.

[2] N. Chomsky, Minimalist Inquiries: The Framework, in Step by step, eds., R. Martin, D. Michaels, and J. Uriagereka, pp. 89-155, MIT Press, Cambridge, MA, 2000.

[3] N. Chomsky, Beyond Explanatory Adequacy, MIT Occasional Papers in Linguistics Number 20, Cambridge, MA, 2001.

[4] K. Johnson, Towards an Etiology of Adjunct Islands, Ms., University of Massachusetts at Amherst, 2002.

[5] G. Grewendorf and J. Kremers, Phases and Cyclicity: Some Problems with Phase Theory, The Linguistic Review, vol. 26, pp. 385-430, 2009.

[6] N. Chomsky, The Minimalist Program, MIT Press, Cambridge, MA, 1995.

[7] D. Pesetsky and E. Torrego, T-to-C Movement: Causes and Consequences, in Ken Hale: A life in Language, ed., M. Kenstowicz, pp. 355-426, MIT Press, Cambridge, MA, 2001.

[8] J. Ginsburg, A Computational Model of Language Generation Applied to English Wh-Questions, In Preparation.

[9] N. Chomsky, Bare phrase structure, in Evolution and Revolution in Linguistic Theory, eds., H. Campos and P. Kempchinsky, pp. 51-109, Georgetown University Press, Washington, DC, 1995.

[10] L. Rizzi, The Fine Structure of the Left Periphery, in Elements of Grammar, ed., L. Haegeman, pp. 281-337, Kluwer, Dordrecht, 1997.

[11] S. Miyagawa, EPP, Scrambling and Wh-in-situ, in Ken Hale: A life in Language, ed., M. Kenstowicz, pp. 293-338, MIT Press, Cambridge, MA, 2001.