

Computation with Probes and Goals: A Parsing Perspective

Sandiway Fong
Departments of Linguistics and Computer Science
University of Arizona
Tucson AZ
USA
sandiway@email.arizona.edu

Abstract

This paper examines issues in parsing architecture for a left-to-right implementation of the probe-goal Case agreement model, a theory in the Minimalist Program (MP). Computation from a parsing perspective imposes special constraints. For example, in left-to-right parsing, the assembly of phrase structure must proceed through elementary tree composition, rather than using using the generative operations MERGE and MOVE directly. On-line processing also poses challenges for the incremental computation of probe/goal relations. We describe an implemented parser that computes these relations and assembles phrase structure, whilst respecting the incremental and left-to-right nature of parsing. The model employs two novel mechanisms, a Move and a Probe box, to implement efficient parsing, without “lookback” or unnecessary search of the derivational history.

1 Introduction

Recently, there has been a shift in the structure of linguistic theories of narrow syntax from abstract systems of declarative rules and principles, e.g. [Chomsky, 1981], to systems where design specifications call for efficient computation within the human language faculty. In particular, recent work in the Minimalist Program, e.g. [Chomsky, 1998,1999], has highlighted the role of locally deterministic computation in the construction of syntactic representation.

Instead of a system involving Spec-Head agreement, Chomsky re-analyzes the Case-agreement system in terms of a system of *probes*, e.g. functional heads like T and v^* , that target and agree with *goals*, e.g. referential and expletive Ns, within their c-command domain. Under this system, probe-goal agreement can be long-distance and may not necessarily trigger movement, e.g. in the case of *there*-expletive constructions and Quirky Case agreement in Icelandic.¹ The implemented parser described in this paper represents the first implementation of the probe-goal account. The logical separation of agreement

¹ In this system, probe-goal agreement may trigger concomitant movement by the principle of Maximize Matching Effects, [Chomsky, 1999].

and movement distinguishes this system from those based on the Minimalist Grammar (MG) formalism [Stabler, 1997]. In the MG formalism, formal feature-checking always precipitates movement.

Efficient assembly, i.e. locally deterministic computation, from a generative perspective with respect to (bottom-up) MERGE does not guarantee that parsing with probes and goals will also be similarly efficient. By *locally deterministic computation*, we mean that the choice of operation to apply to properly continue the derivation is clear and apparent at each step of the computation. In the case where it is not possible to decide between actions, we have a *choice point*. A theory that efficiently assembles phrase structure starting from a primitive lexical array may not have a correspondingly efficient procedure for the left-to-right recovery of that phrase structure since the LA is not available prior to parsing. A simpler example can be used to illustrate the point. There is a well-known, efficient procedure for forming the product r of two prime numbers, p and q . On the other hand, decomposing r into p and q requires a relatively computationally expensive procedure, necessitating guesswork or search.

This paper describe a implemented system that handles a range of examples discussed in [Chomsky, 1998,1999]. In particular, it explores the computational and empirical properties of the probe-goal system from a left-to-right, incremental parsing perspective.

Instead of MERGE and MOVE as the primitive combinatory operations for the assembly of phrase structure, we describe a system driven by elementary tree composition with respect to a range of heads in the extended verb projection (v^* , V, c and T). Elementary tree composition is an operation that is a basic component of Tree-Adjoining Grammars (TAG), [Joshi & Schabes, 1997], and other linguistic theories, e.g. [Di Sciullo, 2002]. The system described here is *on-line* in the sense that once an input element has fulfilled its function, it is discarded, i.e. no longer referenced. To minimize search, there is not only no lookahead, but there can also be no *lookback* in the sense of being able to examine or search the derivational history. Instead, we make use of two novel devices with well-defined properties: a Move Box that encodes the residual properties of CHAINS and theta theory, and a single or current Probe Box to encode structural Case assignment and to approximate the notion of (strong) Phase boundaries. In particular, the restriction to a single Probe Box means that probes cannot “see” past another probe; thereby emulating the Phase Impenetrability Condition (PIC). Limiting the Move Box to operate as a stack will allow nesting but not overlapping movement. A consequence of this is that extraction through the edge of a strong Phase is not longer possible. Examples of parses will be used to illustrate the empirical properties of these computational elements. The system is also *incremental* in the sense that a partial parse is available at all stages of processing. In particular, it extends the derivation to the right in a manner reminiscent of [Phillips, 1995].²

The basic questions explored in this paper are as follows: (1) what are the situations where left-to-right computation pose problems for deterministic computation, (2) what computational elements are necessary to implement the on-line assembly of phrase

² The term “reminiscent” is used here because [Phillips, 1995] pre-dates the probe-goal Case agreement framework discussed here.

structure in an efficient manner, and (3) what are the consequences of eliminating computational choice points introduced by the extra machinery.

2 The Lexicon

We begin with the definition of a lexicon: the heart of the implemented system. Following directly from [Chomsky, 1998, 1999], we assume the parser operates with a system of functional and lexical categories with properties and features, interpretable and uninterpretable, of the form shown in Figure 1 below. The property of selection and uninterpretable feature matching will drive the parsing process. In the course of computation, uninterpretable features belonging to analyzed constituents will be eliminated through probe-goal agreement in a manner to be described in detail in section 6. A (valid) parse is a phrase structure that obeys the selectional properties of the individual lexical items, covers the entire input, and has all uninterpretable features properly valued.

There are five basic types of heads listed in the table:

(1) C

Two types of complementizer are represented here; declarative *c* and *c(wh)* for *Wh*-questions.

(2) T

Two types of tense; T for tensed clauses, and ϕ -incomplete or defective T, represented by T_{ϕ} , for infinitivals.

(3) v

Small *v* comes in three basic flavors: transitive v^* , for verbs like *hit*, unergative $v^\#$, for verbs like *swim* and unaccusative *v*, for verbs like *arrive*. Past participles are also analyzed as instances of *v*.

(4) V

In conjunction with the variety of small *vs*, two basic types of *V* with respect to complement-taking are listed. Transitive and unaccusative *V* select for a complement, but not unergative *V*.

(5) N

We restrict our attention to simple nominals, excluding from discussion complex nominals that select for complements.

Lexical Item (LI)	Properties	Uninterpretable Features		Interpretable Features
		ϕ -features	Other	
v* (transitive)	select(V) spec(select(N)) value(case(acc))	per(P) num(N) gen(G)	(epp)	
v (unaccusative)	select(V)			
v [#] (unergative)	select(V) spect(select(N))			
PRT (passive participle)	select(V)	per(P) num(N)	case(_)	
V (transitive) (unaccusative)	select(N)			
V (unergative)				
V (raising)	select(T _{$\bar{\phi}$})			
T	select(v) value(case(nom))	per(P) num(N) gen(G)	epp	
T _{$\bar{\phi}$}	select(v)	per(P)	epp	
c	select(T)			
C (wh)	select(T)		epp	wh
N (referential)			case(_)	per(P) num(N) gen(G)
N (wh)		wh	case(_)	per(P) num(N) gen(G)
N (expletive)		per(P)		

Figure 1: A Sample Lexicon

The heads $c(\text{wh})$, T , $T_{\bar{\phi}}$, v^* and v are *probes* with uninterpretable features, and participate in the fundamental Agree operation, to be discussed in section 6. The elements, properties and features, of this table are rendered in pseudo-PROLOG notation and are grouped as follows:

(6) Select:

For example, $\text{select}(V)$ is a property of v^* ; that is, v^* selects for a (complement) phrase headed by V . v^* also has the property $\text{spec}(\text{select}(N))$; this notation is used to indicate that v^* pre-selects for a phrase headed by N in its specifier position.³

(7) ϕ -Features:

The structures $\text{per}(_)$, $\text{num}(_)$ and $\text{gen}(_)$ are used to represent the ϕ -features person, number and gender, respectively, with the anonymous logic variable ($_$) representing the uninstantiated slot for the value of each feature. In the case of the probe v^* , these features are uninterpretable (and come unvalued). For nominals, these features are interpretable (and come valued). Probe-goal agreement will value the uninterpretable features, i.e. fill the slots indicated by the anonymous logic variable.

(8) Value:

For example, T has property $\text{value}(\text{case}(\text{nom}))$; that is, T as a probe values nominative Case for an appropriate goal. Similarly, in this system, (transitive) v^* values accusative Case.

Defective T , indicated by $T_{\bar{\phi}}$, differs from T in that it has an incomplete set of ϕ -features (just person $\text{per}(_)$), and cannot value Case (no $\text{value}(\text{case}(_))$ property). Selectionally, they are the same, i.e. they both select for phrases headed by v .

(9) Case:

Nominals will have the uninterpretable feature $\text{case}(_)$ (with an open slot for a value). Through the Agree relation, probes with the property $\text{value}(\text{case}(V))$, where V is nom or acc will instantiate an appropriate slot in a nominal goal, thus eliminating the uninterpretable feature for the goal.

(10) EPP:

The EPP is an uninterpretable feature with a special property. Elements that possess this feature (epp) may trigger MOVE, defined in (13). epp licenses a specifier position as the landing site for movement. If the MOVE operation succeeds, uninterpretable epp is eliminated. Unique among the features introduced here, the EPP feature (or property) can also be satisfied by MERGE. For example, T has feature EPP. It can be eliminated either by raising, say, the internal subject of v^* to specifier- T or by direct merge of an expletive like *there* as in *there is a man in the room*.

³ As will be explained below, $\text{select}(V)$ plays only a role in elementary tree composition. In particular, it is not a participant in the central operations Agree or Move.

(11) **Q and wh:**

We assume that the *Wh*-word fronting system works in a parallel fashion to the Case agreement system. Q, or c(*wh*) here, has interpretable feature *wh*, which cancels with uninterpretable feature *wh* for *Wh*-nominals under Agree.

The lexical definitions given above, along with an appropriate encoding of Agree and Move, suffice to determine basic phrase structure. For example, the parse generated by the system for the simple sentence *John saw Mary* is shown in Figure 2 below. In this case, the displayed features show that *John* is subject to MOVE (to be elaborated on in section 4), receiving nominative Case from T. In return, T's ϕ -features are valued. There is a similar exchange in the case of v^* and *Mary* with respect to accusative Case.

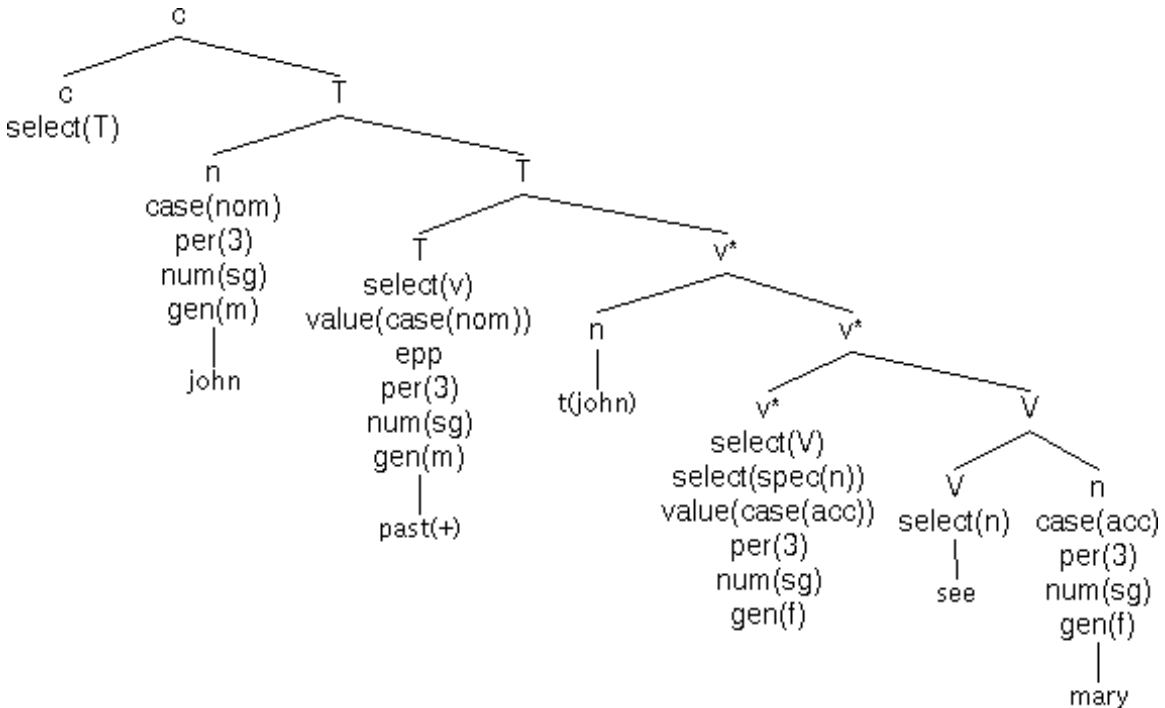


Figure 2: Example parser for *John saw Mary*

3 Elementary Trees

The basic operations MOVE and MERGE, defined in (12) and (13) respectively, are fundamentally bottom-up operations for the assembly of phrase structure. An online, left-to-right parser cannot make use of these operations directly. In this section, we describe an alternative mechanism based on the composition of (possibly underspecified) elementary trees.

$$(12) \quad \text{Merge}(\alpha, \beta) = \{\alpha, \beta\} = \gamma, \text{LB}(\gamma) = \text{LB}(\alpha) \text{ or } \text{LB}(\beta)$$

α , β and γ are syntactic objects. Syntactic objects are either primitive lexical items (LI) or the products of Merge. LB is the label function.

Agree (defined later in section 6) in the presence of EPP triggers Move.

- (13) $\text{Move}(p,g)$ holds if:
- $\text{Agree}(p,g)$ holds, and
 - p has an EPP-feature.
- Then:
- Identify some $\text{PP}(g)$ (pied-piping), and
 - Merge $\text{PP}(g)$ to some specifier- p leaving a trace, and
 - EPP- p is deleted

Probe p and goal g are syntactic objects. g is in the c-command domain of p .

Elementary trees form the base component of Tree-Adjoining Grammars (TAG) [Joshi & Schabes, 1997]. We will assume parsing proceeds (in part) through composition of elementary trees that contain open positions. The range of elementary trees is determined by lexical properties. Given the lexicon of Figure 1, we define the 9 *ground* elementary trees shown in Figure 3 below. By *ground*, we mean that all the sub-components of the tree are defined or specified. (We return to discuss examples of *non-ground* or underspecified trees shortly.)


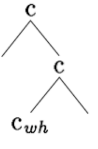
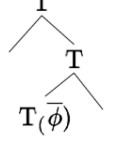
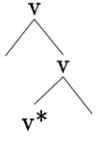

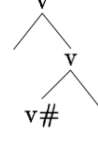

c	c_{wh}	T and $T_{\bar{\phi}}$	Nominal	
			N	
(a)	(b)	(c)	(d)	
$v^*/v/v^\#$			V (unergative)	V (transitive/ unaccusative)
			V	
(e)	(f)	(g)	(h)	(i)

Figure 3: Elementary Trees

Elementary trees are basically projections of functional and lexical heads with open complement and specifier positions pre-determined by lexical entries. For example, the lexicon in Figure 1 defines three versions of v . Both transitive v^* and unergative $v^\#$ have selectional properties $\text{select}(V)$ and $\text{spec}(\text{select}(N))$ represented by elementary trees with two open position, as shown in (e) and (g) in Figure 3. Unaccusative v has selectional

property $\text{select}(V)$ only, so it has just one open position (for its complement). In the case of T and $T_{\bar{\phi}}$, the epp feature translates into an open specifier position, as shown in (c).

With these basic building blocks, it is a straightforward matter to “paste together” or perform elementary tree composition to form a parse tree for a complete sentence such as *John saw Mary* in Figure 2, filling in the open positions on the edge of the tree from the input in a linear, left-to-right fashion. The basic procedure is given in (14) and the sequence of steps to assemble Figure 2 beginning with the complementizer (c) is given in Figure 4.⁴

Step	Phrase Structure	Input
(i)	$[_c c _]$	John saw Mary
(ii)	$[_c c [_T _ [_T T _]]]$	John saw Mary
(iii)	$[_c c [_T \text{John} [_T T _]]]$	saw Mary
(iv)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] _]]]$	see Mary
(v)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v _ [_v [_v v^* _]]]]]]$	see Mary
(vi)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* _]]]]]]$	see Mary
(vii)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* [_v V _]]]]]]]]$	see Mary
(viii)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* [_v [_v \text{see}] _]]]]]]]]$	Mary
(ix)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* [_v [_v \text{see}] \text{Mary}]]]]]]]]$	(empty)

Notes:

(i)	$c:\text{select}(T)$
(ii)	fill specifier-T
(iii)	fill head-T
(iv)	Assume $\text{saw} \rightarrow \text{past}(+) \text{ see}$.
(v)	$T:\text{select}(v)$
(vi)	Insertion of $t(\text{John})$. See section 4.
(vii)	$v:\text{select}(V)$
(viii)	fill head-V
(ix)	fill complement-V

Figure 4: Assembly of *John saw Mary*

(14) **Parse:**

- a. Given a category c , pick an elementary tree headed by c .
- b. From the input:
 - i. Fill in the specifier (if one exists)
 - ii. Fill in the head
 - iii. Fill in the complement by recursively calling parse with c' where c has lexical property $\text{select}(c')$

⁴ In Figure 4, the underscore character ($_$) is used to denote (unfilled) open positions.

Less straightforward is the matter of picking out the right elementary tree each time around the parse procedure. In Chomsky's generative model, assembly begins with a one-time selection from the lexicon that produces a lexical array (LA). In other words, the correct components for assembly are laid out in a separate step ahead of assembly time. In the case of on-line parsing, no pre-determined LA is available. Lexical items associated with the input can only be discovered in the course of assembly. Not knowing the LA forces the introduction of a choice point at elementary tree selection time, e.g. the selection of v^* (over v and $v^\#$) in (v) and transitive/unaccusative V (over unergative V) in (vii) in Figure 4.

We can limit choice point formation in some cases by underspecifying or keeping non-ground parts of the elementary tree. More abstractly, an elementary tree can be *linearly underspecified* with respect to whether it has a complement, e.g. V, and its lexical properties, e.g. T/T_ϕ . An abstract elementary tree can be substituted in these cases and the final shape of the elementary tree determined when the head is inserted (modulo lexical polysemy). In cases where underspecification of the specifier is required, as with v^* versus $v^\#/v$, this strategy will not result in choice point elimination since the (potential) specifier position must be filled before the head in strict left-to-right order.

Summarizing with respect to Figure 3, limited elementary tree underspecification in the implementation permits cases (e) and (g) to be conflated; also cases (h) and (i). With respect to the sequence of steps in Figure 4, underspecification allows (local) determinism to be maintained for steps (ii), selection of T, and (vii), selection of V; but not for steps (i), selection of c, and (v), selection of v, where the option of the specifier position cannot be resolved without the benefit of lookahead.

The fact that v and V are largely decoupled here, in the sense that different variants of v may co-occur with a given V, permits the system to flexibly handle examples of causative/unaccusative alternations such as (15a-b) at the cost of introducing non-determinism.⁵

- (15) a. $[_T \text{ The sun } [_T T [_v t(\text{sun}) [_v v^* [_v \text{ melted the ice }]]]]]]$
 b. $[_T \text{ The ice } [_T T [_v v [_v \text{ melted } t(\text{ice})]]]]]]$
 c. $*[_T \text{ The sun } [_T T [_v v [_v \text{ melted the ice }]]]]]]$

Note that parser cannot detect that (15c), cf. (15b), is illicit until it reaches the verb object position. That is, local determinism in the choice of v cannot be maintained.⁶

⁵ Lexico-semantic constraints external to the system described here will be needed to rule out cases like **John arrived Mary*.

⁶ In Chomsky's bottom-up generative framework, (15c) cannot be assembled. $\text{Agree}(T, \text{ice})$ will force the raising of the object according to the principle of maximizing matching effects, i.e. Agree will trigger MOVE if possible. For the parsing model, as will be explained later, assembly will fail at the verb object position due to constraint (16), i.e. the preference for the Move Box over the input.

4 The Move Box

The Move Box is used by the parser to encode phrasal movement.⁷ The Move Box represents a “holding cell” or a piece of short-term memory that is used to hold constituents that undergo MOVE. Open positions in the parse tree may be filled by the contents of the Move Box. This component of the parser is reminiscent of the *ad hoc* HOLD register used for filler-gap dependencies in Augmented Transition Networks (ATN) [Woods, 1970]. However, the Move Box defined here is simply an embodiment of, and strictly respects, theta theory. In other words, box manipulation is strictly constrained by a small set of operations that encode theta theory as it applies to traditional Chains, encoding the history or derivation of movement.

Initially, let us assume the simplest case of a single Move Box. The introduction of this data structure immediately presents a problem for deterministic computation. We have introduced a choice point; namely, the option of filling an open position from the Move Box instead of the input. Let us eliminate this choice point immediately with the following preference rule:

(16) Move Box Preference Rule

When filling open positions) always prefer the Move Box over the input.

In particular, (16) asserts that, provided the Move Box is non-empty, we must always select from the Move Box, irrespectively of the contents of the input. There is no choice involved. In other words, (16) removes the choice point in step (b) of the (revised) parse procedure, as shown below in (17). (We will return to consider the empirical consequences of this strategy later.)

(17) Parse:

- a. Given a category c , pick an elementary tree headed by c .
- b. From the Move Box *or* input:
 - i. Fill in the specifier (if one exists)
 - ii. Fill in the head
 - iii. Fill in the complement by recursively calling parse with c' where c has lexical property $\text{select}(c')$

We now turn to the operating conditions of the Move Box, i.e. the conditions under which the box may be initialized, filled and emptied. At the start of the parse, the Move Box contains nothing:

(18) Move Box: Initial Contents

Empty.

⁷ We do not consider the computation of affixes and head movement in this paper. The computation of these elements may fall outside the purview of narrow syntax.

Hence, initially, elementary tree open positions are filled from the input. However, whenever an open position is filled from the input, we will make a copy and place it in the Move Box:

(19) Move Box: Fill Condition

When filling from input, copy to Move Box.

As mentioned earlier, the Move Box respects theta theory. In particular, once we arrive at a selected position that needs to be filled, we have essentially determined the original MERGE position of the moved phrase, and the parser’s (re-)construction of the “chain” of movement is complete. As the contents of the Move Box are no longer required by computation, it is deleted:

(20) Move Box: Empty Condition

At a selected position, empty it.

(In this model, the selected positions are the theta positions $\text{spec}(\text{select}(N))$ and $\text{select}(N)$ for v^* and V in Figure 1, respectively.)

Note also that conditions (19) and (20) logically combine to fill and immediately empty the Move Box in the case of *in situ* elements.

We are now in a position to illustrate the operation of the Move Box. Consider again the sequence of operations shown in Figure 4 for the simple sentence *John saw Mary*. The corresponding manipulations for the Move Box are documented in Figure 5 below.

Step	Phrase Structure	Input
(i)	$[_c c -]$	John saw Mary
(iii)	$[_c c [_T \text{John} [_T T -]]]$	saw Mary
(vi)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* -]]]]]]$	see Mary
(ix)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* [_v [_v \text{see}] \text{Mary}]]]]]]]]$	(empty)

Move Box notes:

Step	Move Box	
(i)	(empty)	Initial condition.
(iii)	John	Fill specifier-T from input, copy <i>John</i> to Move Box
(vi)	(empty)	Fill from Move Box, $t(\text{John})$. Empty Move Box. (Specifier- v^* is a selected position.)
(ix)	(empty)	Move Box is empty. Fill from input (<i>Mary</i>).

Figure 5: Move Box computation for *John saw Mary*

Note that the Move Box must be empty at the start of step (ix) in Figure 5, given the Move Box preference rule. However, it is also important that the lifespan of the box be carefully controlled and not, for example, be emptied prematurely. Consider example (21a) and the corresponding parse in (21b). Here, the Move Box containing *prizes* must be available for successive cyclic movement.

- (21) a. Several prizes are likely to be awarded
 b. [_c c [_T several prizes [_T [_T past(-) [_v [_v be] [_A [_A likely] [_T t(prizes) [_T T_φ [_v [_v PRT] [_v award t(prizes)]]]]]]]]]]

There is one further complication that needs to be addressed. To accommodate expletive movement, i.e. the movement of an expletive from one non-selected position to another (possibly iterated), we need to refine condition (20) as follows:

- (22) **Move Box: Empty Condition for Expletives**
 Fill from Move Box at a non-selected position: if box contains an expletive, *optionally* empty it.

Note that we have introduced an (unavoidable) choice point in (22). Emptying is made an option to accommodate the possibility of recursion, as illustrated in (23). With recursion, it is not possible to locally determine whether a given non-selected position is the last or original (MERGE) position of the expletive.

- (23) a. There are prizes awarded⁸
 b. There are likely t(there) to be prizes awarded
 c. There are supposed t(there) to be likely t(there) to be prizes awarded

Making (22) deterministic by not emptying the Move Box in the case of an expletive will also produce incorrect results. For example, in (23b), the Move Box must be emptied otherwise the parser will not be able to pick up *prizes* from the input.

The Move Box preference rule (16) also has certain desirable consequences. Consider again (15c), the case of (incorrectly) selecting unaccusative *v* over transitive *v**, repeated here as (24):

- (24) * [_T The sun [_T T [_v v [_v melted the ice]]]]

(25) summarizes the state of the computation at the point where the parser is poised to complete the verb object position. The parser has not encountered any selected positions, so it must fill from the non-empty Move Box, thereby orphaning or stranding the contents of the input (*the ice*). Hence, (24) is ungrammatical.

- (25) a. [_c c [_T The sun [_T [_T past(+)] [_v v [_v melt _]]]]]]
 b. the ice (Input)
 c. the sun (Move Box)

A Move Box preference also blocks illicit passivization of an indirect object, as in (26):

- (26) *Mary was given a book to t(Mary)

⁸ For the examples in (23), Chomsky assumes an English-particular rule of Thematization/Extraction (TH/EX) at PF will front *prizes* ahead of the verb *award*. This rule is currently unimplemented in the parser described here.

Assuming a small clause-style analysis of the double object construction, e.g. along the lines of [Pesetsky, 1995], the parser must select *Mary* from the Move Box (over *a book*) to fill the specifier-P open position in (27).

- (27) a. [_c [_T Mary [_T [_T past(+)] [_v PRT [_v [_v give] [_p - [_p [_p to] -]]]]]]]]
 b. a book (Input)
 c. Mary (Move Box)

5 Limitations of the Move Box

The single Move Box system has some design limitations. In some cases, as will be discussed in this section, it will become necessary to invent additional boxes. However, for example, with two or more boxes, we will have to choose which one to fill from. Hence, multiple boxes are to be avoided if possible, or at least constrained in a manner that does not promote non-determinism in the system. Organizing boxes into a non-flat data structure such as a stack, i.e. nesting, is an example of a strategy that does not promote non-determinism. The access rules for this data structure are clear, i.e. we can only pick or have access to the (current) top box. No choice is required.

5.1 Nesting

Consider the two cases of *wh*-object extraction in (28a-b):

- (28) a. Who did Bill see?
 b. Who was a book given to?

(28a-b) contain examples of *nested* movement, as shown in (29a-b), respectively. For both cases, *who* occupies the Move Box when the parser reaches the specifier-T (or subject) position. The subject, *Bill* in (28a) and *a book* in (28b), also needs to occupy the Move Box, since it is also part of a (non-trivial) chain, originating in specifier-v and specifier-P, respectively.

- (29) a. Who did [_T Bill [_v t(Bill) [_v v* [_v see t(who)]]]]
 b. Who was [_T a book [_T [_T past(+)] [_v PRT [_v give [_p t(book) [_p to t(who)]]]]]]]

In both cases, the problem can be solved by allowing Move Boxes to be nested by recency, i.e. in stack fashion. The following three rules govern the creation and deletion of multiple boxes:

- (30) **Move Box: Nesting**
 (When filling non-selected open positions) allow filling from the input (creating a new Move Box).

(31) **Move Box Preference Rule**

Operations may only reference the most recently created Move Box

(32) **Move Box: Deletion**

When a Move Box is emptied, it is deleted.

For example, (30) allows *Bill* in (non-selected) specifier-T in (29a) to begin a new Move Box. This second box is the new top-of-stack. By the second preference rule (31), all box operations must now reference this box, thereby eliminating a potential choice point. Proceeding normally, this second box is emptied when *t(Bill)* is inserted in specifier-v (a selected position). At this point, the second box can be discarded, following rule (32), as it has fulfilled its theta duties in the sense that the movement chain is now complete, and the original Move Box containing *who* can be reactivated. Parsing proceeds normally, and this box is subsequently emptied at the verb object position. A similar sequence of actions apply in (29b), with the second box containing *a book* emptied and eliminated at specifier-P.

Finally, note that the parser will still (correctly) reject (26). In state (27), the open position is *not* a non-selected position, and thus a new box cannot be created.

5.2 Overlap

We distinguish nesting from *overlap* with respect to chains. In this paper, we consider all cases of overlap to be undesirable, as it requires more powerful parsing machinery. Consider example (28a) again, repeated below as (33a). In Chomsky's model, heads such as *c* and *v** constitute (largely impenetrable) strong Phases. The Phase Impenetrability Condition (PIC) limits the scope of probes for feature matching. For example, in order for *who* to be visible to the *wh*-probe *c(wh)*, it has to be first extracted to the Object Shift position, an "escape hatch" at the edge of the phase. As can be seen in (33b), this results in movement chain overlap that cannot be accommodated by the machinery described earlier for nesting.

(33) a. Who did Bill see?

b. [_c Who [_c [_c *c(wh)*] did [_T Bill [_v *t(who)*] [_v *t(Bill)*] [_v *v**] [_v see *t(who)*]]]]]]

Both overlap and nesting require multiple boxes. However, in (33b), we require access to two boxes, or the ability to choose between them, thereby compromising determinism. Put another way, overlap requires more powerful machinery (than nesting) in the sense that it introduces an extra choice point. For parsing, as will be described in the next section, on-line, left-to-right processing implies that Agree between *who* and *c(wh)* can be obtained without going beyond strong Phase boundaries. In particular, there is no need for movement to the edge for such cases, and we will obtain much of the force of the Phase model through the architectural limit of a single Probe Box, without having to expand beyond the nesting mechanism.

6 Probes and Goals

In this section, we introduce the notion of a Probe Box. Agree is the central relation computed by the parser. The operation that implements Agree will always involve the participation of a current probe, stored in the Probe Box, with a freshly introduced element of the input. In other words, Agree is performed as early as possible in an on-line fashion.

6.1 Agree and Value

Formally, Agree is defined in (34) in terms of matching features (ϕ -features or *wh*) between active probes and goals. Syntactic objects are *active* if they have one or more (undeleted) uninterpretable features.

- (34) Agree(p, g) if
- Match(p, g) holds. Then:
 - Value(p, g) for matching features, and
 - Value(p, g) for property value(f)

Probe p and goal g are syntactic objects. f is a feature.

Following Chomsky's definitions, if Match(p, g) holds, the goal g may value uninterpretable features in the probe p , indicated by Value(g, p):

- (35) Value(α, β) holds if:
- (Unify) Unify matching ϕ -feature values of α and β .
 - (Assign) If α has property value(f), f for β receives its value from α .

α and β are syntactic objects or features of syntactic objects.

For example, as the parse in Figure 2 indicates, v^* 's (uninterpretable) ϕ -features are valued by matching ϕ -features of *Mary*. Proceeding in the opposite direction, a probe p may value uninterpretable features of a goal g if p has the property of valuing some feature f . For example, T and v^* in Figure 2 have property value(case(nom)) and value(case(acc)), valuing the (uninterpretable) structural Case feature of *John* and *Mary*, respectively. (Note that the ϕ -incomplete probe T_{ϕ} does not have this property and thus cannot value uninterpretable features.)

The model here deviates from Chomsky's basic account in that logical unification is used in (35a) in order to maintain the single Probe Box story in the context of ϕ -incomplete probes, as will be explained below.

6.2 The Probe Box

For parsing, the probe p in (34) will always refer to the current contents of the Probe Box. At the start of the computation, the Probe Box is empty:

(36) **Probe Box: Initial Contents**
Empty.

We modify the parse procedure to call Agree and update or maintain the contents of the Probe Box in an on-line fashion, as shown in (37). The basic strategy is to run Agree on items as soon as they are inserted into phrase structure. With this strategy, no new choice points need be introduced.

(37) **Parse:**

- a. Given a category c , pick an elementary tree headed by c .
- b. From the Move Box *or* input:
 - i. Fill in the specifier (if one exists)
 - ii. Run Agree(p,s) if p and s are non-empty
 - iii. Fill in the head h
 - iv. Run Agree(p,h) for goal h or ϕ -incomplete h
 - v. Copy h to Probe Box p if h is a probe
 - vi. Fill in the complement by recursively calling parse with c' where c has lexical property select(c')

Assuming the current elementary tree contains a specifier s , step (37b-ii) runs Agree(p,s) as soon as the specifier position is filled. Next, as the head of the elementary tree is filled, if it is a probe, it is copied to the Probe Box by step (37b-v), possibly overwriting a pre-existing probe. Only a single probe is permitted. Note step (37b-iv) also stipulates that Agree is also run on heads that are ϕ -incomplete probes. (We return to discuss this operation in the next section.)

We are now in a position to illustrate the operation of the Probe Box for a simple sentence. Consider again the sequence of operations shown in Figure 4 for *John saw Mary*. The corresponding manipulations for the Probe Box are documented in Figure 6 below.

Step	Phrase Structure	Input
(i)	$[_c c -]$	John saw Mary
(iv)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] -]]]]$	see Mary
(vi)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* -]]]]]]]]$	see Mary
(ix)	$[_c c [_T \text{John} [_T [_T \text{past}(+)] [_v t(\text{John}) [_v [_v v^* [_v [_v \text{see}] \text{Mary}]]]]]]]]]]$	(empty)

Probe Box notes:

Step	Probe Box	
(i)	(empty)	Initial condition.
(iv)	$[_T \text{past}(+)]$	Fill head of T from input, $[_T \text{past}(+)]$ is a probe, copy $[_T \text{past}(+)]$ to Probe Box
(vi)	$[_T \text{past}(+)]$	Fill specifier-v from Move Box Apply (37b-ii), run $\text{Agree}([_T \text{past}(+)], \text{John})$. $\text{value}(\phi\text{-John}, \phi\text{-T})$, $\text{value}(\text{T}, \text{Case-John})$.
(vi)	v^*	New head v^* is a probe, copy v^* to Probe Box ($[_T \text{past}(+)]$ is displaced.)
(ix)	v^*	Fill complement-V from input, Apply (37b-iv), run $\text{Agree}(v^*, \text{Mary})$. $\text{value}(\phi\text{-Mary}, \phi\text{-}v^*)$, $\text{value}(v^*, \text{Case-Mary})$.

Figure 6: Probe Box computation for *John saw Mary*

The Probe Box is filled by $[_T \text{past}(+)]$ in step (iv). In step (vi), Agree is carried out on specifier-v *John*. The ϕ -features of *John* value the ϕ -features of the current probe T, and T values the Case feature of *John*. The single Probe Box strategy implies that the new probe v^* displaces T as the current probe. In step (ix), Agree is carried out on complement-V. The ϕ -features of *Mary* value the ϕ -features of v^* , and v^* values the Case feature of *Mary*.

The single Probe Box model incorporates and preserves much of the property of Phases with respect to locality. A probe cannot penetrate into the domain of a lower probe since it will be displaced as soon as the parser encounters the second probe.

6.3 ϕ -Incomplete Probes

So far, the current probe has been determined by left-to-right parse order. Let us now turn to situations of the kind considered by Chomsky involving intervening ϕ -incomplete probes such as infinitival $T_{\bar{\phi}}$ and PRT, shown here in (38)-(40).

(38) a. We expect there to arrive a man

b. We T expect $[_T \text{there } T_{\bar{\phi}} \text{arrive a man}]$

(39) a. There are likely to be awarded several prizes

b. There T are likely $[_T t(\text{there}) T_{\bar{\phi}} [_v \text{PRT} [_v \text{award several prizes}]]]$

- (40) a. There are expected to arrive a man
 b. There T [_v PRT [_v expect [_T t(there) T_φ arrive a man]]]

There are two problems to be addressed in examples (38)-(40). (1) the presence of ϕ -incomplete probes blocking matrix T from agreeing with the object of the embedded clause, and (2) the valuation of the uninterpretable features belonging to ϕ -incomplete probes.

Matrix T must value the Case feature of the embedded object and its ϕ -features must be valued by the embedded object. Under the single probe model, matrix T must not be displaced by T_φ or PRT, since the contents of the Probe Box must be preserved until the parser reaches the embedded object position and can run Agree. This is encoded in (41):

- (41) **Probe Box: ϕ -Incomplete Probes**
 ϕ -incomplete probes may not occupy the Probe Box.

However, (41) by itself is insufficient since ϕ -incomplete probes also have uninterpretable features that must be valued for the computation to succeed. With respect to the lexicon defined earlier in Figure 1, T_φ has a single uninterpretable ϕ -feature *person*(), and the participle PRT has uninterpretable features {*person*(),*number*()}. Assuming feature unification, the solution to this problem is given in parsing step (37b-iv). When a head *h* that is also a ϕ -incomplete probe is analyzed, *Agree*(*p,h*) is applied. For example (38), *Agree*(T, T_φ) will unify the (still) unvalued features of T and T_φ. In particular, T-*person*() is unified with T_φ-*person*(). More precisely, the feature value slots, denoted by “”, are unified and will share values once instantiated. T stays in the Probe Box and goes on to complete *Agree*(T,*man*). The uninterpretable ϕ -features of T are valued. Furthermore, T_φ-*person*() is also valued by association from the earlier unification. Similar considerations apply for T_φ and PRT for (39) and (40). In the case of PRT, in addition to its ϕ -features, it also has an uninterpretable Case feature, which is valued by *Agree*(T,PRT). Hence, the single Probe Box can be maintained and no additional choice points need be created.⁹

6.4 Probe Inactivation

In the account so far, the Probe Box is only superseded when a new (ϕ -complete) probe comes along to fill the box. In particular, a probe may remain in the Probe Box even after its uninterpretable ϕ -features have been valued. In other words, the Probe Box may hold an inactive probe. In this section, we give evidence that this strategy is essentially correct. Consider example (21a) again, repeated below as (42a). (b) encodes the state in

⁹ Actually, we also need to add that the parser runs *Agree*(*c*(*wh*),*wh*-N) locally, i.e. when the *wh*-N is first filled at specifier-*c*, to maintain the single Probe Box story for *wh*-movement.

computation where the embedded specifier-T has just been filled with *prizes* from the Move Box:

- (42) a. Several prizes are likely to be awarded
 b. [c c [T several prizes [T [T past(-) [v [v be][_A [_A likely][_T t(prizes) [T T_φ –]]]]]]
 c. prizes (Move Box)
 d. T (Probe Box)

In accordance with applying Agree as soon as possible, Agree(T, *prizes*) at step (42b) will value T's ϕ -features (and the Case feature of *prizes*). Without outstanding uninterpretable features, T is rendered inactive. However, as (43) indicates, it is necessary to allow T to remain in the Probe Box as there are still ϕ -incomplete probes, namely T_φ and PRT, that need to have uninterpretable features valued through Agree with T.

- (43) [c c [T several prizes [T [T past(-) [v [v be][_A [_A likely][_T t(prizes) [T T_φ [v [v PRT][_v award t(prizes)]]]]]]]]]]

7 A Preliminary Comparison

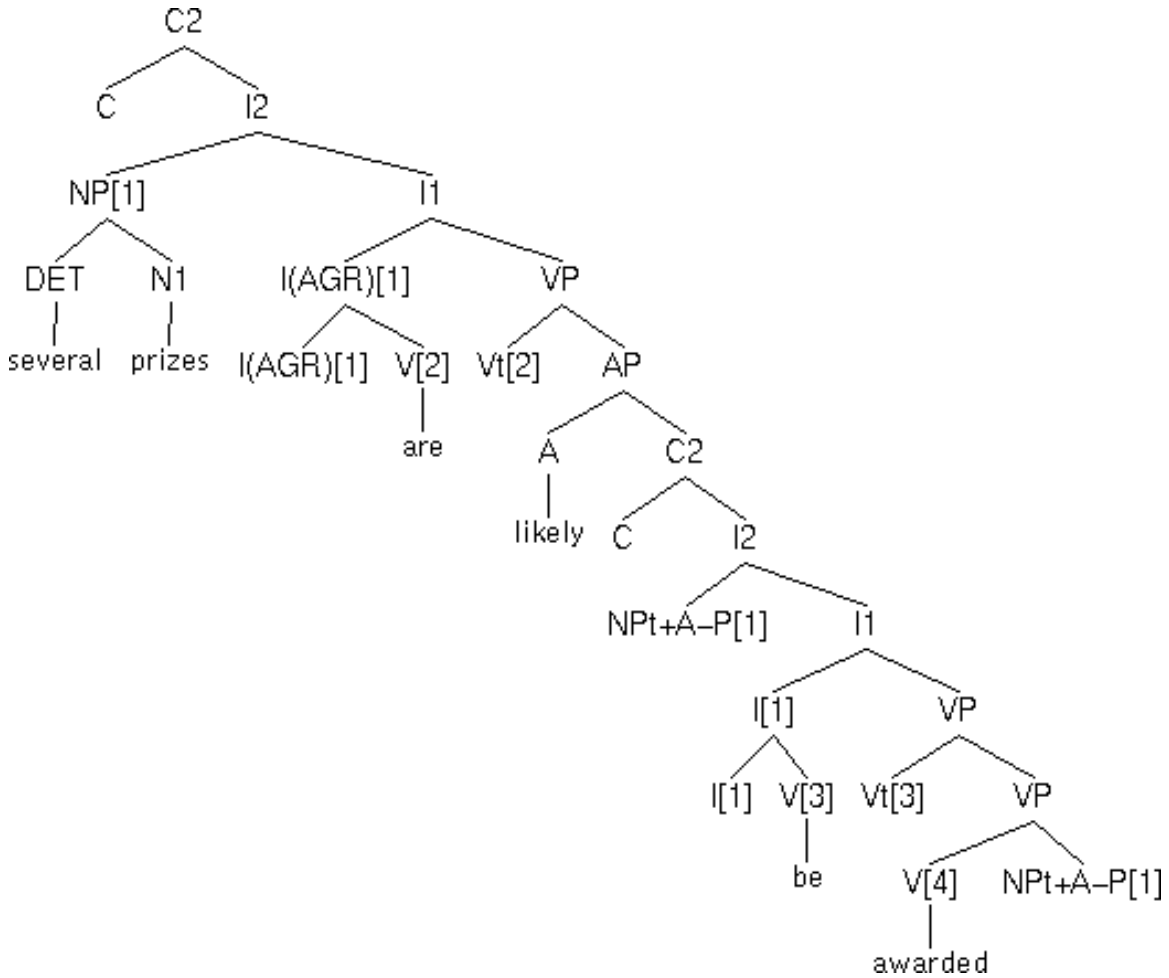
In this section, we compare the number of computational steps taken by the probe-goal parser to that taken by a corresponding parser PAPPi [Fong, 1991] in the Government-and-Binding (GB) framework [Chomsky, 1981], for the analysis of examples (44a-b):

- (44) a. There are likely to be awarded several prizes
 b. Several prizes are likely to be awarded

We should point out that the results reported in Figures 7 and 8 are preliminary. The parses given in Figures 7 and 8 are for example (44b) for the GB-based and probe-goal parsers, respectively. Although the parses recovered are similar to one another, the linguistic coverage of the two parsers are quite different. Currently, the GB-based parser has much wider coverage, and carries more overhead in terms of computational machinery, thereby affecting the results to some degree. However, even with this caveat in mind, the difference in computational efficiency between the probe-goal parser and the GB-based parser is noteworthy. This is reflected both in terms of the amount of structure built and the number of movement operations during parsing. In the former case, the GB-based parser constructs approximately an order of magnitude more syntactic objects than its probe-goal counterpart. (The reported results are normalized in terms of elementary trees units (eT).¹⁰) With respect to movement, there is a similar order of magnitude difference. One reason for this striking difference is that the GB-based parser is designed around a generate-and-test model of computation, where declarative principles interact and freely combine both to provide for and limit the range of possible parses without

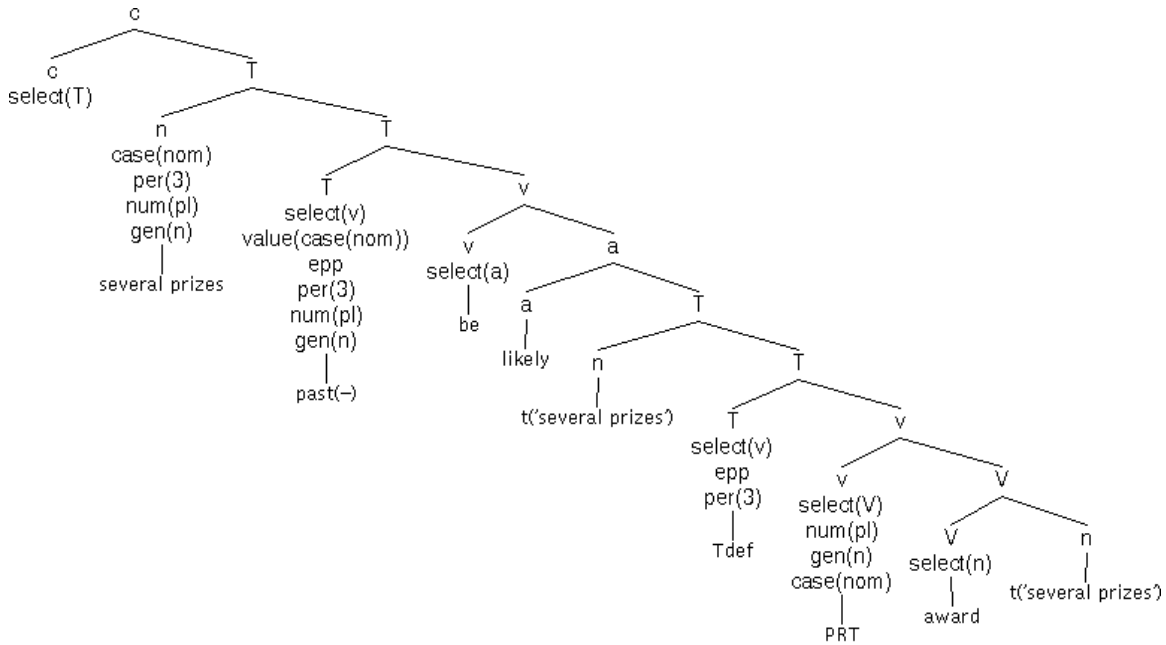
¹⁰ The GB-based parser builds structures by composing individual phrases rather than in elementary tree-sized chunks. The exchange rate is roughly 5-to-1 since each fully populated elementary tree contains 5 syntactic objects.

regard for linear order. By contrast, the probe-goal parser has been designed around a more constrained model where linguistic constraints strictly follow the order imposed by left-to-right, on-line computation.



Example	Structure Building	Move- α
(44a)	1864 LR \cong 373 eT	26
(44b)	1432 LR \cong 286 eT	67

Figure 7: PAPI parse for *Several prizes are likely to be awarded*



Example	Structure Building	Agree/Move
(44a)	15 eT	5/2
(44b)	20 eT	7/7

Figure 8: Probe-goal parse for *Several prizes are likely to be awarded*

8 Conclusions

This paper has outlined a parsing-centric view of computation with probes and goals. The use of elementary tree fragments instead of MERGE and MOVE follows from left-to-right parsing constraints.

Two data structures, the Move and Probe boxes, have been introduced to encode theta-theory and probe-goal locality, respectively. At any given point, the two boxes carry forward deeper into computation syntactic objects that must still interact with other objects not yet parsed. The short-term or “cache” memory represented by the boxes obviate the need to perform *lookback*, i.e. a search back into the computational history for appropriate matching elements. No lookback is a constraint imposed by the commitment to on-line processing. To avoid unnecessary search whilst allowing movement sequences to nest, the Move Box follows a stack organization. The Probe Box is able to hold onto tighter bounds; it maintains its singularity through probe-goal unification in the case of ϕ -incomplete intermediate probes.

Finally, preliminary investigations suggest that following through on these design elements may result in more efficient computational systems, as compared to earlier theories. Further work is required to determine whether this can be maintained as the linguistic coverage of the probe-goal system expands.

Acknowledgements

This work has been supported in part by NEC Laboratories America. The author is indebted to Roger Martin for his help in getting started on this project. Parts of this paper have also been presented at the CUNY Psycholinguistics Supper Club. The author is also grateful for comments received there.

References

- Chomsky, N. A. *Lecture on Government and Binding*. Foris Publications, 1981.
- Chomsky, N. A. *Minimalist Inquiries: The Framework*, MITWPL, 1998.
- Chomsky, N. A. *Derivation by Phase*. MITWPL, 1999.
- Di Sciullo, A.-M. The Asymmetry of Morphology. In *Many Morphologies*. Ed. Boucher, P. Cascadilla Press, 2002.
- Fong, S. *Computational Properties of Principle-Based Grammatical Theories*. Ph.D thesis. Artificial Intelligence Laboratory. MIT, 1991.
- Joshi, A. and Y. Schabes. Tree-Adjoining Grammars. In *Handbook of Formal Languages, vol 3*. Eds. Rosenberg, G. and A. Salomaa. pages 69-123. Springer-Verlag, 1997.
- Pesetsky, D. M. *Zero Syntax: Experiencers and Cascades*. MIT Press, 1995.
- Phillips, C. *Order and Structure*. Ph.D thesis. MIT, 1995.
- Stabler, E. P. Jr. Derivation Minimalism. In *Logical Aspects of Computational Linguistics*. Ed. Retoré, C. pages 68-95. Springer, 1997.
- Woods, W. A. *Transition Network Grammars for Natural Language Analysis*. Communications of the Association for Computing Machinery. 13(10), 1970.