Computational Intelligence, Volume 16, Number 3, 2000

PARSING AND INTERPRETATION IN THE MINIMALIST PARADIGM

JAMES S. WILLIAMS AND JUGAL K. KALITA*
Department of Computer Science, University of Colorado

In this paper, we discuss how recent theoretical linguistic research focusing on the Minimalist Program (**MP**) (Cho95, Mar95, Zwa94) can be used to guide the parsing of a useful range of natural language sentences and the building of a logical representation in a principles-based manner. We discuss the components of the MP and give an example derivation. We then propose parsing algorithms that recreate the derivation structure starting with a lexicon and the surface form of a sentence. Given the approximated derivation structure, MP principles are applied to generate a logical form, which leads to linguistically based algorithms for determining possible meanings for sentences that are ambiguous due to quantifier scope.

Key words: Natural language understanding, Minimalist Program, Principles-based parsing.

1. INTRODUCTION

In this paper, we introduce a framework for describing the grammar of natural languages due to Noam Chomsky called the Minimalist Program (MP). We investigate how to build a parser that produces syntax trees conforming to the MP and how aspects of language that have bearing on meaning, but cannot be conveniently captured during parsing, can be processed. The linguistic framework is discussed first, followed by the computational implementation.

We first give a brief overview of the Minimalist Program (Chomsky 1995; Marantz 1995; Zwart 1994). The MP is the latest incarnation of Principles and Parameters grammars that define language structure in terms of well-motivated principles and parameters that adapt the principles to various natural languages. However, in this paper, we do not describe a parser that implements the MP in all its cognitive implications, which are still not completely understood and are being investigated. We feel that it is a worthwhile exercise to use the basic principles of the MP to obtain rules and structures that enable the traditional implementation of a parser. Parsing, however, is only a part of the processing. There are many linguistic phenomena than can be handled only after a syntax tree has been obtained. We discuss, among other issues, further processing of the parse to handle issues in quantifier scoping. We think this part of the paper is interesting since it shows how vexing linguistic phenomena can be handled using simple computational techniques.

In writing the parser, we use a set of sentence types that have been considered by those who have written parsers motivated by earlier versions of Principles and Parameters grammars. Merlo, in his paper on a parser based on an earlier version of Principles and Parameters grammar, called the Government and Binding Theory (GB), wrote that the set of sentences he chose constitutes a

> "crucial test set for principle-based parsers: (they involve) complex interactions of principles over large portions of the tree." (Merlo 1995, pp. 521–522).

* Address correspondence to J. K. Kalita, Department of Computer Science, University of Colorado, Colorado Springs, CO 80917; e-mail: kalita@pikespeak.uccs.edu

The sentence types that Merlo covered and we also handle are as follows:

(1) **Simple Transitive:** John loves Mary.
(2) **Simple Intransitive:** John runs.
(3) **Simple Passive:** Mary was loved.
(4) **Simple Raising:** Mary seems to like John.
(5) **Embedded Transitive:** John thinks that Mary loves Bill.
(6) **Embedded Intransitive:** John thinks that Mary runs.
(7) **Embedded Raising:** Mary thinks that John seems to like Bill.
(8) **Simple Question:** Who does John love?
(9) **Embedded Question:** Who do you think that John likes?
(10) **Embedded Question and Raising:** Who did you think that John seemed to like?
(11) **Embedded Wh-Question:** *Who did you wonder why Mary liked?

Of course, Merlo's assertion can be questioned. However, we believe that a parser that can handle Merlo's sentences and more in light of the MP can be considered an important step in building so-called principles-based parsers. Later in the paper we consider additional sentence types to illustrate the problems in interpretation that are introduced by the use of quantifiers such as *some* and *every* either in simple sentences or in the context of raising verbs such as *seem* and *appear*. Examples of some such sentences are given below.

(12) Someone attended every seminar.
(13) Everyone loves someone.
(14) Someone seemed to attend every class.
(15a) Someone persuaded John to attend every class.
(15b) Someone hoped to recite every poem.
(15c) Someone believes John to be attending every class.
(16a) Someone expected every Democrat to win.
(16b) Someone expected every Democrat would win.
(17) Everybody beat his donkey.

We extend a GB theory grammar to work with MP for parsing sentences and labeling arguments according to positions in chains. We describe chain-building algorithms necessary for approximating the MP derivation structures. Then, in Section 4 we discuss algorithms to build logical representation from approximate derivational structures based on Horstein 1995. In Section 5, we describe the program we have built, in Section 6 we compare it with other principle-based parsers, and in Section 7 we present our conclusions.

2. THE MINIMALIST PROGRAM

The Minimalist Program (Chomsky 1995) generalizes principles developed within the Government and Binding Theory (Haegeman 1994; Chomsky 1981; Culicover 1997) so that derivations of sentences can be explained using the idea of least effort or least work. The MP and GB both explain language in terms of a set of principles called the *Universal Grammar* (UG) and a set of parameters fixed during language acquisition.

It is assumed that the Conceptual-Intentional (CI) system in the human brain enables meaning comprehension and that the Cognitive System deals with sentence formation. The Cognitive System produces the so-called Phonetic Form (PF) or the surface word order with pronunciation details and passes it to the part of the brain

that pronounces the sentence. The interface between the Cognitive System and CI system is called Logical Form (LF). The LF provides additional information that helps in interpretation.

We are concerned here with the Cognitive System (Figure 1). The information that the Cognitive System works with is termed Lexical Resources (LR). The Computational System (CS) within the Cognitive System combines the Lexical Resources to derive a sentence. A derivation begins with the LR and at a point called *Spell-Out* splits and heads toward the PF and LF interfaces. Movements prior to Spell-Out are called *overt movements* because they can affect word order in the spoken sentence; those after Spell-Out are *covert movements* because they are not observed in speech but are required for grammaticality reasons. The area between LR and Spell-Out is the *Working Area* (Marantz 1995, pp. 360–361). After the derivation splits at Spell-Out, one copy of the derivation goes to the PF interface and is pronounced. The other copy undergoes further movements until it reaches the LF interface that is needed for semantic reasons. The order of words at Spell-Out is the surface word order for the language. Different languages have different surface word order characteristics because the overt movements that occur in the languages are different even though they all start in the same initial configuration.

2.1. Lexical Resources

An item in LR is composed of several types of features (see Table 1). Phonological features indicate how to pronounce words. Semantic features indicate meanings of words and are used by the Conceptual-Intentional system. Formal features, the only ones used within the Compuational System, include categorial, ϕ, and case features. *Categorial features* are $\pm$N (nominal) and $\pm$V (verbal) (Haegeman 1994). These essentially tell us the grammatical category of a lexical item. The ϕ-*features* are person, number, and gender; they are features of an overt expression. We assume that Lexical Resource items have a case feature even if it is not overtly marked. Case features include nominative (the subject or actor case), accusative (the object or affected case), and genetive (the possessive case). In English, case is not overtly marked except in the case of pronouns.

Items in LR are of two types: *lexical* and *functional*. Lexical items correspond to heads of phrases like NP and VP. Functional items are determined by principles of UG and parameterization. Lexical and functional items occur in the syntactic tree as it is built during the derivation of a sentence. Functional items are used to provide targets where lexical items move during the process of derivation to meet grammaticality requirements.

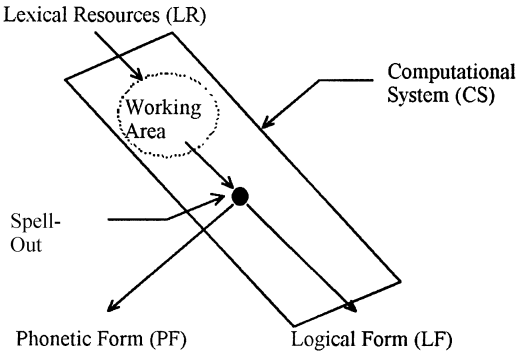

FIGURE 1. Components of the Cognitive System.

TABLE 1. Formal Features for Lexical Items

Lexical item	Formal features
Mary	+N, −V (NP) Case: accusative ϕ (3rd person, singular, feminine)
John	+N, −V (NP) Case: nominative ϕ (3rd person, singular, masculine)
loves	−N, +V (verb) Tense: present ϕ (3rd person, singular, masculine) Assign case: accusative ϕ (3rd person, singular, feminine)

For example, in order to derive sentence (1) we place the lexical items *Mary, John*, and *loves* into the Working Area. We follow (Allen 1995) and assume that NPs can be identified reliably. The CS attempts many derivations with a range of lexical items. Most derivations crash (i.e., they fail) or do not result in the intended meaning. Among the derivations that succeed, the one that is most economical is the grammatical derivation of a sentence. The economy criteria for derivation have been stated informally for the MP.

Formal features other than categorial features are used for *checking*. Checking between two items refers to making sure one or more features on the two items agree. The many such checks that are performed during the derivation process ensure that the derived sentence is grammatical.

Checking matches identical features in two possible scenarios: a Head-Head or a Spec-Head relation. A *head* of a phrase is the primary node of the phrase that defines its essential characteristics. *Head-Head checking* is done when two heads are adjoined. It is possible that during the process of derivation one head moves and attaches to another head in a special manner called *adjoining*.

Spec-Head checking is done when features match between a head and its specifier. A phrase can have a node called the *specifier* that provides an argument to the head. For example, in the sentence *John loves Mary*, *John* is the subject of *loves*. This fact can be represented by making the node *John* the specifier of the head node *loves* in a VP phrase.

When paired features are checked they become "invisible" (Chomsky 1995, p. 229). The lexical item *loves* has two sets of ϕ-features. The first is for checking subject-verb agreement, the second for object-verb agreement. In other words, one set of ϕ-features is checked at a functional node (named Agr_sP) that ensures subject-verb compatibility, and another set of ϕ-features is checked at another functional node (named Agr_oP) that ensures object-verb compatibility.

We also require the functional items in Table 2 to be put in the Working Area. Agr is an abbreviation for agreement, T for tense, and C for complementizer. As we have seen already, there are two types of Agr: Agr_s stands for Subject agreement and Agr_o stands for Object agreement.

A functional item has many features. Two sets of features that are relevant at this time are N-features and V-features. N-features of a functional head (such as Agr_s) are

TABLE 2. Formal Features for Functional Items

Functional item	Formal features
Agr_s	Weak N-features[ϕ(3rd person, singular, masculine)] Weak V-features[ϕ(3rd person, singular, masculine)]
Agr_o	Weak N-features[ϕ(3rd person, singular, feminine)] Weak V-features[ϕ(3rd person, singular, feminine)]
T	Strong N-features[case: nominative] Weak V-features[tense: present]
C	

a bundle of features that are used, during derivation, to check agreement between the functional head and its nominal argument. V-features are a set of features that are used to check agreement between the functional head and the verb. For example, subject-verb agreement in English (or, any language) is ensured using a two-step process. One step matches the verb's features with the functional head called Agr_s. Another step matches the subject NP's features with the same functional head Agr_s. The order in which these two steps are carried out is a parameter of the language. Additionally, whether the two steps are performed before or after Spell-Out is also language dependent. Features that are checked before Spell-Out are called *strong features* because they trigger overt movements of components in the derivation tree. Features that are checked after Spell-Out are called *weak features*. These features need to be checked for grammaticality.

Which N-features and V-features are strong and which are weak is a parameter that determines word order for a language. For English, the strengths of features are shown in Table 3.

Movement of items is a fundamental activity of all Principles and Parameters theories of grammar. These theories work by requiring that a lexical item move from its initial location in the derivation structure tree to another location (specifically, a functional item) to check features. N-features are checked against NPs in association with one of the two Agr nodes, and V-features are checked against verbs on Agr nodes. As noted earlier, for convergence, strong features must be checked prior to Spell-Out; weak features are checked after Spell-Out. Designating some features as strong and others as weak is the way the principle of *Procrastinate*, one of the main theoretical are pillars of the MP, is implemented. Procrastinate is "a principle that prefers derivations that hold off on movements until after Spell-Out..." (Marantz 1995, p. 357). Movements before Spell-Out may change surface word order and are considered more expensive compared to movements after Spell-Out that do not have outward effect. Since the MP prefers derivations with lower cost, the MP prefers that items that can wait to

TABLE 3. Strength of Functional Item Features in English

Functional item features	English
Agr N-features	Weak
Agr V-features	Weak
T N-features	Strong
T V-features	Weak

move until after Spell-Out must wait for movement. Convergence at the PF interface overrides Procrastinate and forces some movement prior to Spell-Out. See Zwart (1994, pp. 8–9), Marantz (1995, p. 357, 371–373), and Chomsky (1995, pp. 197–199).

2.1.1. Working Area and the Derivation of a Sentence. We now discuss how an example sentence is derived in the MP. We start with a set of LR items and perform movements to produce a derivation that succeeds (i.e., *converges*) at the PF and LF interfaces. For a set of items, the one derivation, among the many possible, that converges with the least effort is grammatical. Our description of the derivation follows those by Chomsky (1995), Zwart (1994), and Marantz (1995).

Before we give a step-by-step description, let us look at the basic structure of a phrase. The MP follows the tradition of the GB and uses only binary branching trees. This is called the X-bar theory (Haegeman 1994; Culicover 1997). Figure 2 shows the basic structure of an X-Bar phrase used in the MP. In this figure, X stands for any lexical or functional category such as N, V, Agr_0, and T. Therefore, we can instantiate the phrase tree as NP, VP, or TP. X′ is the first level projection of X, and XP is the phrase level projection of X. A phrase tree for any phrase category X may have two important components: a specifier and a complement in addition to the head. The head X is at the bottom of the phrase tree. The specifier attaches at the top level of a phrase on its left in the case of English. The complement attaches to the lower level projection X′. The specifier and the complement both provide arguments of the head X.

For our example we pick the verb *loves* and project a V′ with an empty complement that we fill with the NP *Mary*. If a constituent has an empty complement projected, the MP requires that it be immediately filled with a constituent from the Working Area (called *generalized transformation*) or with a constituent from within the projected constituent itself (called *singulary transformation*).

We next project V′ to VP with an empty specifier and perform another generalized transformation to substitute the empty specifier with the NP *John*. The lexical items in the specifier and complement positions of a verb must move from these initial positions to check or license their features (Zwart 1994, p. 1).

Agr_o is selected next and projected to Agr'_o with a complement position filled by the VP. Agr_o is a functional head that ensures there is agreement between a verb and its object. Such object-verb agreement is not required in a language like English, but there are languages such as Hindi where verbs and objects must agree in form (in some situations) (Mohanan 1994). Agr'_o is projected to Agr_oP. Due to weak N-features, *Mary* will not move to [Spec, Agr_o] prior to Spell-Out, so the specifier is not projected.

Next, the T is selected and projected to T′ with Agr_oP as the complement. The functional head called T contains a specification of a tense that is required by the verb in a sentence; the actual verb later moves to T to make sure it has the same tense

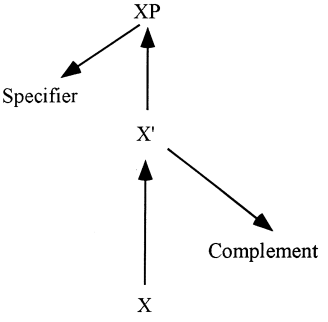

FIGURE 2. Basic X-Bar phrase structure.

as is required of the sentence. T′ is projected to TP with no specifier (Marantz 1995, p. 367). Agr_s is selected next and projected to Agr'_s with TP as the complement. Now, T raises and adjoins to Agr_s since T's N-features are strong and have to be checked before Spell-Out. Adjoining is shown as a "+." A head moving and adjoining to another head is called *head movement*. Head movement creates a chain with two links. If the two heads had features in common they could be checked now. However, the reason T is raised to Agr is due to T's strong N-feature, case, that must be checked before Spell-Out (Marantz 1995, p. 367).

Next, Agr'_s is projected to Agr_sP with the NP *John*, a constituent from within the constituent being projected, as the specifier. This is a singulary transformation. The movement from the [Spec, V] to [Spec, Agr_s] occurs due to a principle called *Shortest Move* which states that, "a constituent must move to the first position of the right kind up from its source position" (Marantz 1995, p. 355). For an explanation see Chomsky (1995, pp. 181–186). This movement forms an *A-chain*. A chain is a set of nodes linked together. Argument positions in a phrase tree can be classified into two types: A-positions and A′-positions. A-positions hold arguments that can be assigned special relations called θ- or thematic roles. A-positions of a transitive verb are its specifier and complement; these are usually available as [Spec, Agr_s] and [Spec, Agr_o] respectively. Informally, θ-roles are also called case roles. Examples are roles such as AGENT, PATIENT, and RECIPIENT between a verb and its arguments. A chain's landing site (i.e., the node where the constituent finally lands) can be an A-position or an A′-position. Therefore, chains can also be called A-chains or A′-chains based on where the moved constituent finally ends up.

Movement within the MP follows a *Copy Theory* where a constituent is copied from position to position. Now NP *John*'s N-features can be checked with the adjoined heads. Formally, "A maximal projection *a* agrees with a (functional) head *b* only if *a* is a specifier of *b*" (Zwart 1994, pp. 4–5). Of course, there are some restrictions on what can occupy the specifier position of a phrase.

Finally, C is projected to C′ with an empty complement that is filled with the Agr_sP. The functional head called C is necessary to provide a location where complement phrases in complex sentences can be parked. For example, *that* is a complementizer which can reside in a C node. A complementizer allows us a mechanism to embed a whole sentence within a sentence. See our example sentences, (5), (6), (7), (9), (10), and (11). C′ is projected to CP without a specifier. We have reached the point of Spell-Out. The derivation splits with one copy being submitted to the PF interface. All strong features have been checked and all items in the working area have been transformed into one derivational structure.

2.1.2. Spell-Out to Logical Form. Movements from here on check weak features. V raises and adjoins to Agr_o to check its V-features with head-to-head checking. Formally, "A head *a* agrees with a (functional) head *b* only if *a* is adjoined to *b*" (Zwart 1994, pp. 4–5).

Agr_o and the adjoined V have weak N-features that must also be checked, so [Spec, Agr_o] is projected and the nearest constituent *Mary* is moved. The accusative case of V is checked with *Mary* and the weak N-features of Agr_o are also checked with Mary. The "$V + Agr_o$" complex is raised to "$Agr_s + T$" complex so that V can check its features with the weak V-features of T and Agr_s. Thus, we see that it is possible to raise a complex of nodes. All movements have occurred and the derivation converges at both the PF and LF. Figure 4 sums up the derivation in a layout that saves space. In the figure, solid lines indicate overt transformations to check strong features. Dashed lines indicate covert transformations to check weak features. Italicized words represent the position in a grammatical sentence, the ordering at Spell-Out.

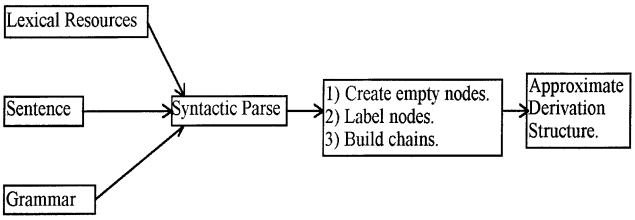

FIGURE 3. Approximating a derivation structure.

3. PARSING

We now describe how to parse sentences in the spirit of the MP. Instead of deriving a sentence from LR, we recreate the derivational structure at Spell-Out by obtaining a traditional parse of the sentence and then extending the syntactic tree by building chains. In the previous section we described how lexical and functional items are combined to form a derivation structure. At Spell-Out the derivation structure is copied and one copy is sent to the PF interface and the other copy continues toward the LF interface. The copy going to the PF interface is pronounced.

Our objective is to approximate the derivation structure given the sentence, the LR, and a grammar. There are two steps to recreating the derivational structure. They are labeled 1 and 2 in Figure 5. Step 1 is to recreate the structure at the point of Spell-Out by determining the overt movements that have occurred. Step 2 is to extend the derivation from the point of Spell-Out to the LF interface by determining the covert movements. Step 3 is needed to obtain the logical form and is discussed a little later.

There are four substeps in the process of recreating overt movements. Following these steps will give us an approximate derivational structure at Spell-Out (see Figure 3).

1. Obtain a syntactic parse for the input sentence.
2. Create empty nodes.
3. Label nodes.
4. Build overt chains.

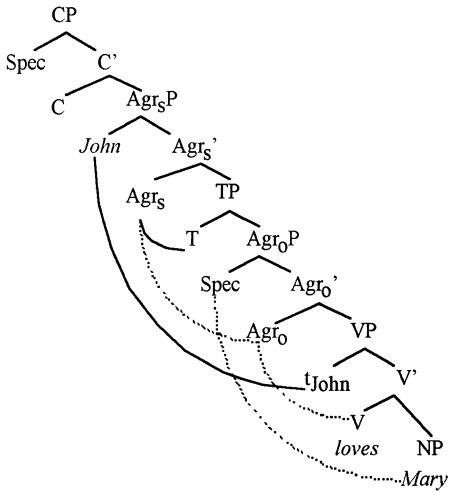

FIGURE 4. The final derivation of *John loves Mary*.

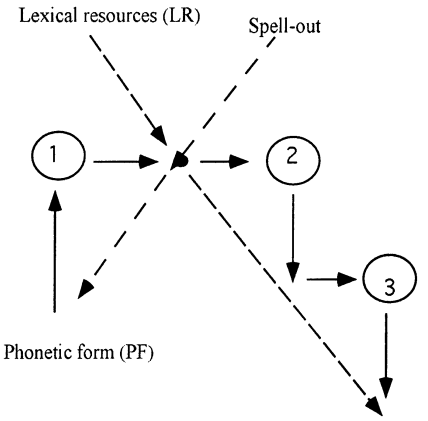

FIGURE 5. Steps to create the derivation structure at the LF interface.

The first step parses the sentence. For our implementation, we use Allen's bottom-up parser described in Allen (1995, Chapts. 3–5). Based on our knowledge of movements in the MP we determine how arguments moved from base positions to the surface positions. We do this by labeling A- and A′-positions, creating empty nodes, and then performing a procedure that builds chains. A′-positions are positions in the phrase structure tree that contain arguments that cannot have θ- or thematic roles. Chain building maps the arguments back to their base VP internal positions.

Our grammar also creates empty nodes. An empty node in the syntactic tree represents a position from which an argument has moved. In the MP, the movement of an argument actually copies the argument to a new position at a higher level in the tree. To recreate these positions, since they are not represented with words in the surface sentence, we have rules that are associated with the grammar rules. This is one way in which we have extended a simple syntactic structure in order to approximate a derivation structure.

The grammar has been designed to cover the sentences described earlier. A parser builds a syntactic tree for a sentence. When a rule is matched, the corresponding Node Labeling rule is invoked to label nodes and create empty nodes. Finally, an algorithm is applied to the syntactic structure to build chains. These chains represent the overt movements (A and A′) that occurred prior to Spell-Out as well as the covert movements (A) that occurred after Spell-Out. The base position of a chain shows the relationship between the argument represented by the chain and the verb. Since verbs assign thematic or θ- roles to their arguments in base positions, we can determine the thematic roles of the subjects and objects.

3.1. Grammar

The grammar in Table 4 was derived in part from Merlo's grammar 3 (Merlo 1995, p. 541). It has been adapted for the MP and to parse the example sentences. Operations that create empty nodes and label argument nodes are listed under Node Labeling Operations. We allow only noun phrases where clauses could be. For instance, in the [Spec, C] position we allow only an NP. We have also explicitly listed rules for transitive, intransitive, and raising verbs. Although this is not strictly necessary, it simplifies empty node creation, node labeling, and chain building.

TABLE 4. An English Grammar Based on the MP

No.	LHS	RHS	Node labeling operations
R_1	S	Agr_s^2	
R_2	S	C^2	
R_3	C^2	N^2 C^1	Label N^2 as A′H
R_4	C^2	C^1	Create empty node to C^1's left; label A′I
R_5	C^1	C Agr_s^2	
R_6	C^1	Agr_s^2	
R_7	Agr_s^2	N^2 Agr_s^1	Label N^2 as AH
R_8	Agr_s^2	Agr_s^1	Create empty node to Agr_s^1's left; label AI
R_9	Agr_s^1	Agr_s T^2	
R_{10}	Agr_s^1	T^2	
R_{11}	T^2	Agr_o^2	
R_{12}	T^2	V_{int}^2	
R_{13}	T^2	V_{rais}^2	
R_{14}	Agr_o^2	Agr_o^1	Create empty node to Agr_o^1's left; label AH
R_{15}	Agr_o^1	Agr_o V_{tran}^2	
R_{16}	Agr_o^1	V_{tran}^2	
R_{17}	V_{int}^2	V_{int}^1	Create empty node to V_{int}^1's left; label AF
R_{18}	V_{tran}^2	V_{tran}^1	Create empty node to V_1^{tran}'s left; label AF
R_{19}	V_{rais}^2	V_{rais}^1	
R_{20}	V_{int}^1	V_{int} C^2	
R_{21}	V_{int}^1	V_{int}	
R_{22}	V_{tran}^1	V_{tran} N^2	Label N^2 as AF
R_{23}	V_{tran}^1	V_{tran}	Create empty node to V_{tran}'s right; label A′F
R_{24}	V_{rais}^1	V_{rais} Agr_s^2	
R_{25}	N^2	N	

The notation used in the grammar is in a form that is slightly different from the linguistic notations used earlier. A maximal projection (i.e., the phrase node such as NP, VP, Agr_oP, etc.) is shown here as the head with a "2" as the suffix. A bar node (usually, a node with not a bar, as in $\bar{N}$ or $\bar{P}$, but a prime such as N′, P′, V′, etc.) is represented here as the head with a "1" as the suffix. Head nodes are represented as the head (Agr_s) or a head followed by a "0". The specifier of Agr_s, [Spec, Agr_s], is a noun phrase represented by N^2, as in the rule (R_7). Empty elements are inserted as a position from which an explicit element may have moved.

A syntactic parse following the grammar for the simple transitive sentence *John loves Mary* is thus:

$$[S\,[Agr_s^2\,[N^2\,[N\ \textit{John}]]\ [Agr_s^1[T^2\,[Agr_o^2\,[Agr_o^1\,[V_{\textit{tran}}^2[V_{\textit{tran}}^1\,[V_{\textit{tran}}\ \textit{loves}]\ [N^2[N\ \textit{Mary}]]]]]]]]]].$$

The brackets indicate the tree structure that corresponds to the parse.

This syntactic structure is different from the derivation structure. For example, the [Spec, V] position has not been projected since there is no sentence element there. Our empty node creation, node labeling, and chain-building rules help us determine when to create an empty A- or A′-node and link that node with a chain to other nodes in the sentence. The head nodes without a corresponding element in the sentence are not created by the grammar.

Let us next consider the parse for the sentence *Who do you think that John seemed to like?*, is given in Figure 6. It is a sentence that has an embedded wh- question and an embedded raising verb *seem*. If we compare its syntactic structure with its derivation structure, we will see several differences. As in the previous example, [Spec, V] is not projected in the syntactic structure. Moreover, [Spec, Agr_s] in the lowest clause is not

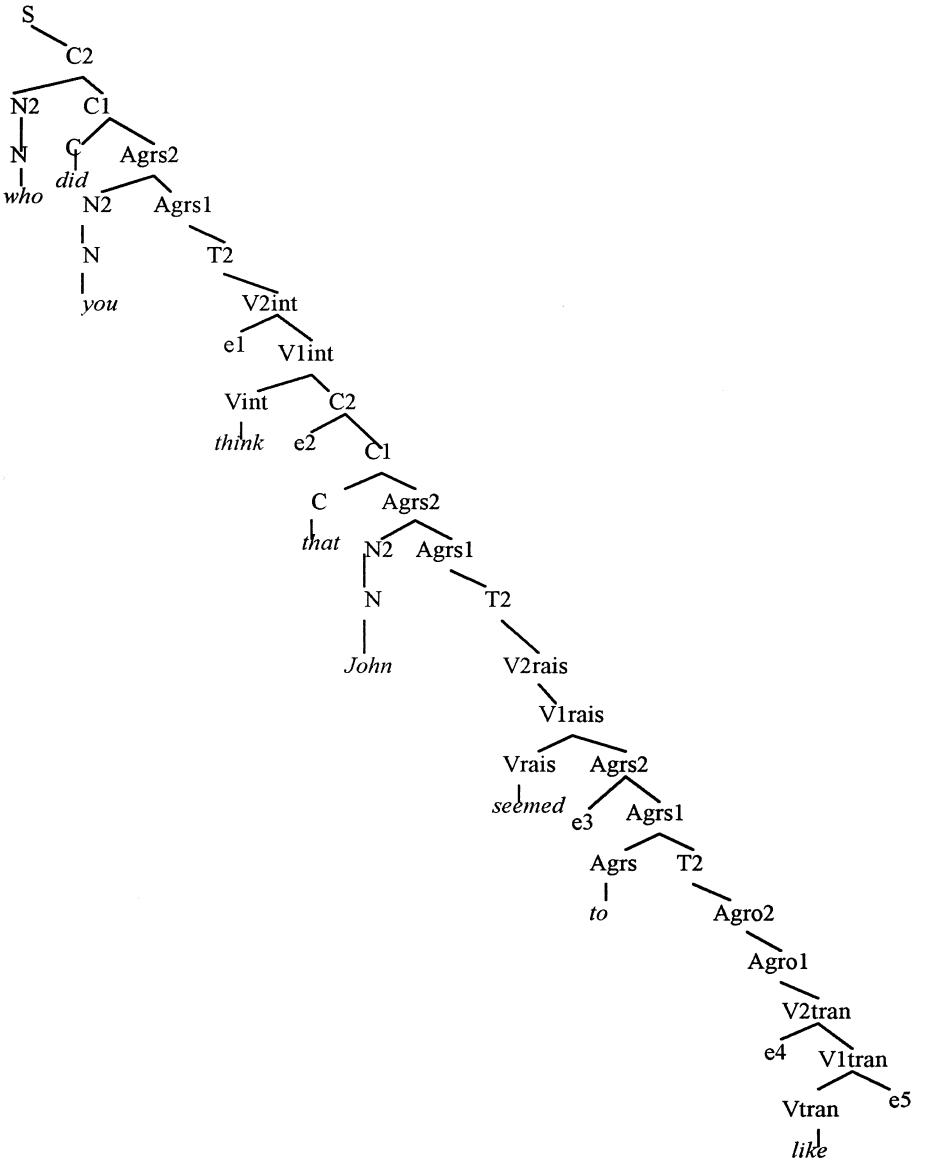

FIGURE 6. Parse for *Who did you think that John seemed to like?*

projected since the subject has risen to the higher [Spec, Agr_s]. The specifier of C, [Spec, C], of the lower clause is also not projected in the syntactic structure. The complement of the transitive verb is also not projected for the same reason. However, transitive verbs require objects. The argument that fills the object of the transitive verb is a chain. The derivation structure is an extension of the syntactic structure.

One question that can be posed at this time is why our grammar rules do not cover the use of expletives such as *there* and *it*. Expletives are of various kinds. Some have case and ϕ-features and others do not. Members of the first type need to have features checked and do not allow raising of their nominal associate. The nominal associate of the expletive *there* is the noun phrase *a book* in the sentence *There is a book on the shelf*. The second type includes "pure expletives," such as *it* and *there* in English, that do not have case and ϕ-features. This latter type does not need checking of features and allows raising of the nominal associate (Chomsky 1995). We will not go into a lot of detail here, for expletives are a complex topic. Even Chomsky says "development of the theory of expletives requires much more careful examination" (p. 289). Considering the difficulties involved in the study of expletives, we decided to ignore them completely in our paper. However, we note that rule R7 in our grammar labels expletives as heads of A-chains. This is not the right treatment for expletives. This is an unintended side effect, but we refrained from correcting the error since the current paper is already very long and if we attempt to cover expletives, the explanation of the relevant rules will make it longer.

We also note that rule R24 allows ungrammatical sentences such as *seems John to escape*. Such sentences can be disallowed by implementing what is called the Case Filter (Haegeman 1994, p. 34; Culicover 1997). A Case Filter module can be easily added to our grammar to rectify the situation.

3.2. Empty Node Creation

There are several rules in the grammar where empty argument positions (nodes) are created. Empty nodes are created in positions that are based on observations of the set of exemplar sentences. For example, in Rule R_4, an *empty node* is created to the left of C^1. This rule reflects that a derivation can have projected [Spec, C] position from which an argument moved. This is an intermediate A′-position for a wh+ phrase. A wh-phrase corresponds to a phrase containing a wh-word such as *whom*, *which*, and *where*. In Rule R_8, a specifier position can also be projected to the left of Agr_s^1. This position is a subject position that is normally filled at Spell-Out. When this rule applies it can be due to an infinitival just below Agr_s^1 and a raising verb above. Since an infinitival is considered to have no tense, the N-features of T will not check case. Therefore, the subject raises to the next higher clause to get its case checked. Rules R_{17} and R_{18} are similar. Both the subject of transitive and intransitive clauses have their base positions within the VP. Due to strong N-features of T in English the subject always moves out of the VP by Spell-Out (discussed later). Therefore this position will never be filled at Spell-Out in English. We create the empty position in rules R_{17} and R_{18} to give us the base position for the chain containing the subject. Rule R_{23} applies when the object has moved prior to Spell-Out. This happens to check strong wh+ features. Figure 6 shows the syntactic tree for one of the sentences discussed in Section 2. The *empty* nodes are labeled e_1 to e_5.

3.3. Node Labeling

There are three kinds of movement: head, A, and A′. The latter two movements are discussed here. In *A-movement* an argument in an A-position moves to another A-position. An A-position is the subject, object, or object of a preposition. *A′-movement*

presents the movement of an argument to an A′-position, which in our discussion will be [Spec, C].

3.3.1. A-Movement. A-movements can occur for different combinations of strong and weak features. N- and V-features are associated with Agr and T and can be strong or weak resulting in sixteen combinations. However, only the N-features cause NPs to move. Strong N-features cause movement prior to Spell-Out and weak N-features after Spell-Out. So there are actually four possible combinations of features that affect NP-movement (A-movement). These are given in Table 5.

Of these four possibilities, only (2) holds for English and therefore, we will discuss only its ramifications. For better understanding, the reader is referred to one of the papers or texts on the Minimalist Program (Chomsky 1995; Marantz 1995; Zwart 1994; Culicover 1997) or to (Williams 1997).

Combination (2) with strong N-features of T causes T to raise and adjoin to the next highest Agr to check those features. N-features are checked in a Spec-Head relationship and, due to Chomsky (1995), [Spec, T] is not projected (Marantz 1995, p. 367). The Agr above T in a derivation tree is named Agr_s. This means that the subject but not the object will move to [Spec, Agr_s] prior to Spell-Out to check the strong N-features of T. Therefore, we can expect subject NPs to be in checking positions at Spell-Out. The [Spec, Agr_s] position is generally filled. If the position is not filled, it can be due to an NP continuing movement in order to check strong features. An example of this is an infinitival clause that does not have case as a strong N-feature of T and the argument moves to a higher clause to check its case. Thus, a [Spec, Agr_s] position will either have an NP and be labeled AH or be empty and therefore labeled AI. We can expect the base position of the subject [Spec, V] to always be empty. Objects are expected to be in base positions since weak N-features of Agr will not force them to move prior to Spell-Out. English is a language with this combination of T and Agr N-features.

3.3.2. A′-Movement. A′-movement has to do with the strength or weakness of C′s +wh features. Following Marantz (1995, p. 378), these are assumed to be strong for English. With this assumption, and following the minimalist principle *Shortest Move*, the closest element with +wh features will move to check the strong features.

A question that might arise at this point is: if +wh is a strong feature, how can one explain why the sentence *Who brought what*? is good? Although, following authors such as Marantz (1995), we assume +wh to be a strong feature, complexities arise in the treatment of strong features in general. Lasnik (1999) states that the means by which strong features determine movements (in the context of wh-movement and other situations) can be discussed by considering several competing theories such as the PF Crash Theory (Chomsky 1993), the LF Crash Theory (Chomsky 1994) and the Virus Theory (Chomsky 1995, pp. 219–394). Why the sentence *Who brought what*? is good will depend

TABLE 5. Possible N-Feature Strengths in Natural Languages

	T N-features	Agr N-features
(1)	Weak	Weak
(2)	Strong	Weak
(3)	Weak	Strong
(4)	Strong	Strong

on what theory is used and some other considerations. One such additional consideration is whether the C (complementizer) node is created in the derivation before or after PF. The discussion is outside the scope of our paper.

3.3.3. Labels. As a sentence is parsed, each A- and A′-position is assigned a label with two letters in it: the first indicating whether it is an A- or A′-chain, the second: letter indicating where it is in a chain—on the top, middle, or bottom. If the position is an A-position, the label has an A prefix. For A′-positions the node label has an A′ prefix. If the node is a head node of a chain, the label has a suffix of H. For intermediate and foot nodes the labels are suffixed with I and F, respectively. To summarize, one of the three labels AH, AI, and AF is associated with each node in an A-chain. Correspondingly, an A′-chain's nodes are labeled with one of the three labels A′H, A′I, and A′F. A chain consists of one or more nodes. The first node, the head, is labeled AH (or A′H). If a chain has two nodes then the first is the head and the second is the foot, AF (or A′F). If a chain has three or more nodes then all the intermediate nodes are labeled AI (or A′I).

3.3.4. Building Overt Chains. Chain building occurs after building the syntactic parse and labeling the nodes. The parse is scanned left to right with the labels on nodes indicating their positions within a chain. If a node is labeled as a head, AH or A′H, then a new chain is started. If the node is intermediate, AI or A′I, then the node is added to the chain on the top of the stack. If the node is a foot then we complete the chain. For our example English sentences, the chains at the point of Spell-Out are nested so that we can use stacks to keep track of the partially built chains. We have a stack for A-chains and a separate stack for A′-chains. The chain-building algorithm is given in Figure 7. The syntactic parse tree for the sentence *Who did you think that John seemed to like?* with labels is given in Figure 8. Table 6 shows the steps to build chains for our example sentence *Who did you* e_1 *think* e_2 *that John seemed* e_3 *to* e_4 *like* e_5*?* Merlo gives an example of chain building for the same sentence (Merlo 1995, p. 532).

3.3.5. Recreating Covert Movements. We now extend the derivation from Spell-Out toward the LF interface by building covert chains. For English and our set of example

```
If node is labeled AH, or A'H Then
  Create a new chain and push it on a stack.
  (StackA for A-chains, StackA' for A'-chains).
Else If node is labeled AI or A'I Then
  Pop partial chain off of appropriate stack
  Insert node into chain
  Push chain back onto stack
Else (nodes are AF or A'F)
  Pop partial chain off of appropriate stack
  Insert node into chain
End If
```

FIGURE 7. A chain-building algorithm.

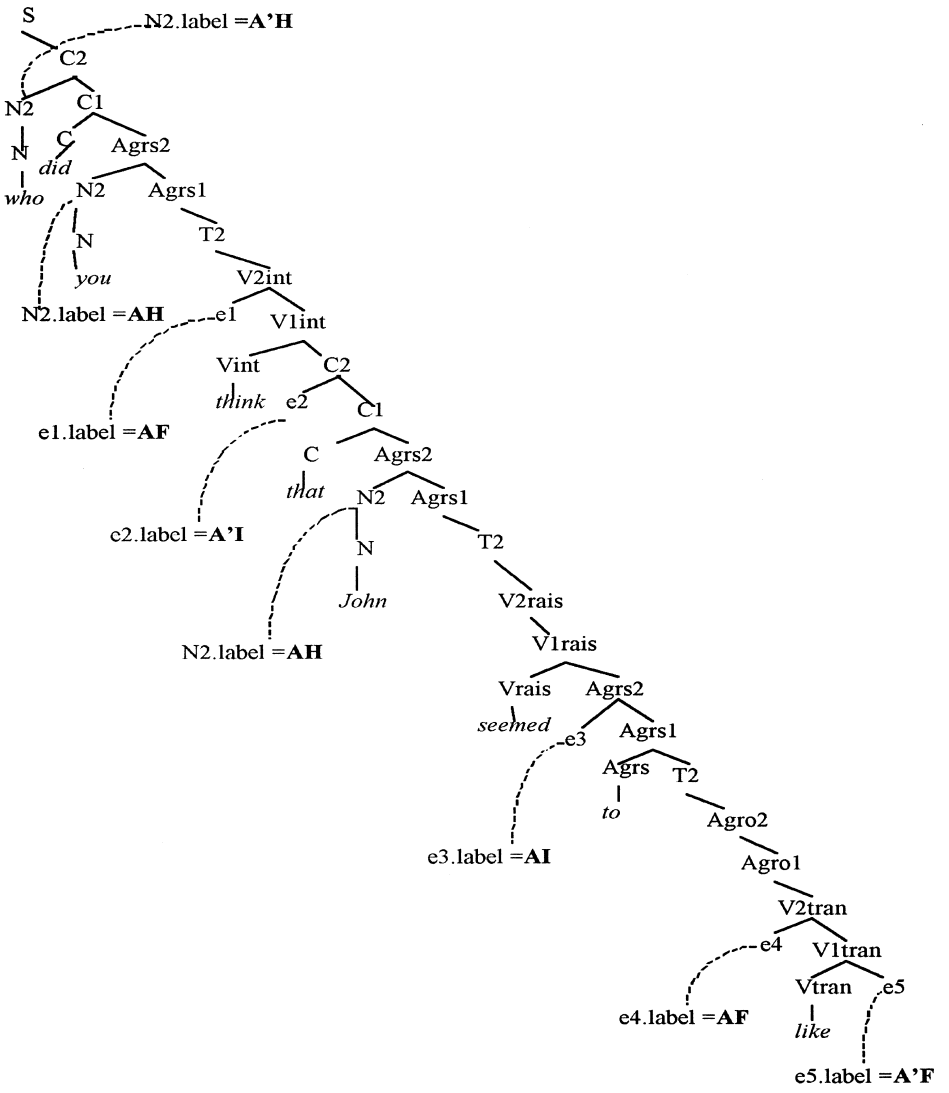

FIGURE 8. Parse with labels for *Who did you think that John seemed to like?*

TABLE 6. Chain-Building Steps for the Sentence *Who did you* e_1 *think* e_2 *that John seemed* e_3 *to* e_4 *like* e_5*?*

Word or empty	Grammar rule	Label	Stack A	Stack A′	Resulting chains
who	$C^2 \rightarrow N^2\ C^1$	A'H	Empty	[who]	
you	$Agr_s^2 \rightarrow N^2\ Agr_s^1$	AH	[you]	[who]	
e_1	$V_{int}^2 \rightarrow (e_1)\ V_{int}^1$	AF	Empty	[who]	[you, e_1]
e_2	$C^2 \rightarrow (e_2)\ C^1$	A′I	Empty	[who, e_2]	
John	$Agr_s^2 \rightarrow N^2\ Agr_s^1$	AH	[John]	[who, e_2]	
e_3	$Agr_s^2 \rightarrow (e_3)\ Agr_s^1$	AI	[John, e_3]	[who, e_2]	
e_4	$V_{tran}^2 \rightarrow (e_4)\ V_{tran}^1$	AF	Empty	[who, e_2]	[John, e_3, e_4]
e_5	$V_{tran}^1 \rightarrow V_{tran}^0\ (e_5)$	A′F	Empty	empty	[who, e_2, e_5]

sentences, this is straightforward. A covert movement takes place when a transitive verb has an object in a base position at Spell-Out. Following the *Shortest Move* principle, the object moves to [Spec, Agr_o].

4. GENERATING LOGICAL FORM

In linguistics, the term *Logical Form* (LF) is characterized as a level of syntactic representation at which all grammatical structure relevant to semantic interpretation is presented (Hornstein 1995, p. 3). It is also stated as the level at which logical properties of a sentence are represented (Culicover 1997). In particular, various phenomena that have impact on meaning such as quantifier scope, scope of negation, pronoun binding, etc., are represented explicitly at LF. The LF structure is the one that is input to semantic interpretation procedures.

In this section, we describe algorithms that take the derivations produced in the previous section and interpret logically at the Conceptual-Intentional interface. One problem is that the chains that we have built have more than one link. In order to converge at the CI interface "any member of an A-chain can be deleted and all but one must be" (Horstein 1995, p. 154). Convergence at the CI interface signifies that a syntactic structure is legal. This requirement is derived from satisfying the Minimalist principle of *Full Interpretation* which "excludes the presence of uninterpretable material at the interface representations" (Zwart 1994, p. 7). In a chain there are multiple nodes containing the same syntactic content, and for us to obtain the meaning of the sentence, only one such node in a chain can be fully interpreted. Otherwise, it will lead to semantic incoherence. This requirement is restated as assumption A_1 later in this section.

The sentences that we have discussed so far are straightforward in terms of selecting an element from each chain to keep. For these sentences, any element will suffice because the meaning representation for a noun phrase used will not affect the meaning of another noun phrase from another chain. This is because they contain no quantifiers. For example, if we look carefully at sentences (1) through (11) listed in Section 1, we see that the noun phrases used are of two types. The first type consists of proper names or referential noun phrases: *John*, *Mary*, and *Bill*. The second type consists of *wh*-words such as *who*. The first type creates no problems of scope because each one is a proper name and no matter which instance of it is chosen from a chain, there is no ambiguity regarding the extent of its scope. A proper noun phrase, which can be considered a constant, has scope everywhere in the sentence. The *wh*-word such as *who* also create no problems. Older versions of Principles and Parameters grammars required *wh*-words to be handled in a separate manner using a process called *wh-movement* before getting to LF, but there is no such need when we consider *wh*-words in terms of the Minimalist paradigm (Hornstein 1995, Chap. 8).

However, for sentences that contain quantifiers, the relative order of the elements does matter and therefore the element selected to remain in each chain matters. For example, in a sentence such as (12), given below, there is ambiguity because we have two quantifiers, *someone* and *everyone*. Quantifiers take scope over a certain domain and they can affect the meaning of other elements in that domain. Depending on which of these two quantifiers is assumed to have wider scope, there are two readings of the sentence. Our goal in this section is to algorithmically determine different possible interpretations of sentences that are ambiguous due to this phenomenon called relative quantifier scoping. We cannot just randomly select one member of each chain to keep and then remove the rest. For example, for the simple transitive sentence (12) we have the structure given in Figure 9.

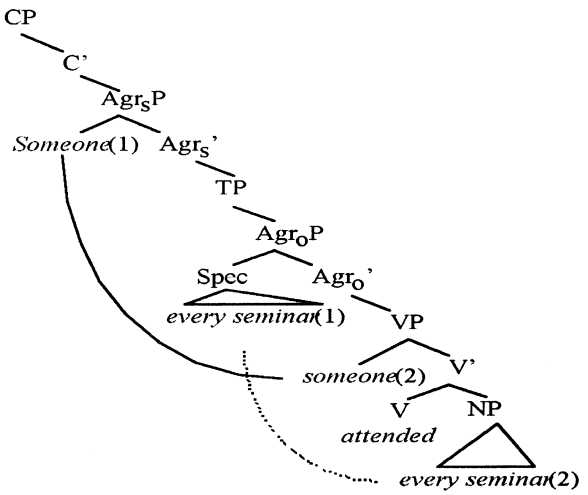

FIGURE 9. Possible positions for *someone* and *every seminar* in sentence (12).

(12) Someone attended every seminar.

There are two interpretations of sentence (12). The first interpretation has a meaning corresponding to: For every seminar x, there is some person y that attended x. The second interpretation has a meaning corresponding to: There is some person y, such that for every seminar x, it is the case that y attended x.

In this sentence, we have two kinds of quantifiers. An *indefinite quantifier* has more than one alternative for referents. For example, *someone* and *a* refer to a single entity but neither one is specific as to which entity. *Definite quantifiers* have a single referent. For example, *everyone* refers to every person within a single context. Another definite quantifier is *the* as in *the seminar*. By the context we know the referent.

The ambiguity in sentence (12) lies in the relative scoping of the arguments. The first interpretation has the universally quantified argument, *every seminar*, with wide scope and it has the existentially quantified argument, *someone*, with narrow scope. The second interpretation has *someone* with wide scope and *everyone* with narrow scope.

4.1. Quantifier Scoping Algorithm

Determining the scope of quantifiers when there is more than one quantifier in a sentence is a difficult problem that has generated copious research. In older versions of Principles and Parameters grammar, such as Government and Binding Theory, the approach to handling quantifiers has been to use *Quantifier Raising* (QR), a process in which quantifiers are forced to move at LF to the top of the phrase structure and attach to a newly formed node (Haegeman 1994; Culicover 1997). If we have several quantifiers in a sentence, all the quantifiers have to move from where they are to the top of the syntactic tree and attach (the technical term is *adjoin*) to newly built nodes on top. The quantifiers can move in various orders and, depending on the order of movement, there are different scopes of quantifiers. A quantifier takes scope over a quantifier below it.

The need for QR is obviated in the Minimalist Program. The movements that have taken place during derivation leading to LF are all the movements that are needed. However, since we have chains in the syntactic tree at LF, and a chain has several nodes with the same content, we must delete all but one node from each chain at LF,

as noted earlier. The choice of nodes to delete gives us the ambiguities in scope that we see for quantifiers.

In linguistics literature, we find several approaches to handling the problem of determining quantifier scope in the context of the Minimalist Program. Some such approaches are due to Hornstein (1995), Szabolcsi (1997) and Beghelli and Stowell (1997). Noting that different quantifiers behave differently in terms of scoping, Szablocsi (1997) subcategorizes quantifiers according to their scoping behavior, thereby distinguishing them syntactically. She extends the representation of quantifiers by adding features that identify their scoping properties and semantic properties as well. Then she postulates new functional heads where noun phrases with determiners have to move to be able to satisfy or check grammatical requirements. This is an approach researchers have taken to solve many linguistic problems in the MP paradigm. However, others, such as Hornstein, take issues with the proliferation of functional nodes. It is Hornstein's opinion that the introduction of such new functional categories is quite often satisfactory from the point of view of derivation but is not well-supported by empirical data. Although this claim is disputed by others, we found Hornstein's explanation of many LF phenomena quite simple and elegant and decided to base our implementation on his ideas. In what follows, we discuss his approach briefly and discuss how we implement it.

It needs to be noted that Stroik (1996) also presents an explanation of scopal properties of quantifiers and some other syntactic phenomena using a modified version of the Scope Theory developed by Aoun and Li (1993). Stroik casts Aoun and Li's theory in the Minimalist framework; however, since we found Hornstein's approach sufficient for our work, we did not pursue Stroik's approach. Moreover, out of the three theories presented here, Hornstein's seemed to be the simplest one to follow.

4.2. Hornstein's Approach to Quantifiers

Hornstein (1995) states the following five assumptions in order to reanalyze quantifier scoping in minimalist terms:

A_1: At the CI interface an A-chain has at most one and at least one lexical link.
A_2: A quantified argument Q_1 takes scope over a quantified argument Q_2 iff Q_1 c-commands Q_2 (and Q_2 does not c-command Q_1).
A_3: A definite argument must be outside the VP shell at the CI interface.
A_4: NPs in English begin in VP internal positions and move out of the VP shell to [Spec, Agr] positions for case checking.
A_5: Movement is copying and deletion.

We have already discussed assumption A_1 earlier in this section. We have also seen A_4 in practice during the derivation of the sentences we discussed earlier. NPs, subjects or objects, start from inside the VP shell and move out either to [Spec, Agr_oP] if it is an object NP or to [Spec, Agr_sP] if it is a subject NP. The movement is needed for feature checking which can be done only in the context of a specifier and its head in the Minimalist framework. However, A_3 imposes a stricter requirement that has impact on quantifier scoping. Hornstein states this as a requirement following arguments made by Diesing (1992). The linguistic discussions are outside the scope of this paper. The other assumptions in the list above will be discussed at the appropriate points below.

4.3. Implementation of Hornstein's Ideas

The algorithm depends on the chains that we formed in the last section following assumption A_4. In Figure 9, *someone* can be in the [Spec, Agr_s] position or [Spec, V]

position. Similarly, *every seminar* can be in the [Spec, Agr_oP] or the V complement positions. All our examples here and in earlier sections have a *c-command* structure in which an element on the left c-commands all the elements on the right in the sentence that follows A_2. The structural relation of c-command among elements of a syntax tree is quite simple. This relation is used by linguists to specify conditions for the occurrence or non-occurrence of many syntactic phenomena.

> "We say that α c-commands β if α does not dominate β and every γ that dominates α dominates β." [Chomsky95:35]

For example, if α is *someone(1)* and β is every *seminar(1)* then *someone(1)* c-commands every *seminar(1)* since every node γ (Agr_sP, C′, or CP) that dominates *someone(1)* also dominates *every seminar(1)*. The leftmost quantifier in the sentence will have wide scope with elements to the right within the domain of the leftmost quantifier.

The algorithm to determine the relative ordering of the elements has steps 3A, 3B, and 3C.

1. Step 3A: Delete all definite elements that are within the VP Shell.
2. Step 3B: For each chain that has adjacent links without an intervening link from another chain, delete all but one of the adjacent links.
3. Step 3C: Show all combinations of selecting one element from each chain and removing all other elements.

In Section 3, we saw that elements moved from one position to another position to check features. The positions an element moved through are represented by the chain. The movement in the Minimalist Program is a copy and deletion theory. The element is copied from position to position and then at the interfaces all but one element in the chain are deleted (Marantz 1995, p. 373).

The VP shell has [Spec, V] and the V complement argument positions. By using the following algorithm for step 3A, we satisfy A_3 and A_5 and we partially satisfy A_1.

Algorithm for step 3A

```
For each definite element in the derivation Do
   If element is in a [Spec, V] or V complement
      position Then
      Delete element
   End If
End For
```

For our example (12), this step removes *every seminar(2)* in the V complement position. This is shown in Figure 10.

In Step 3B, we are concerned with the relative ordering of elements from different chains. Therefore, adjacent elements from the same chain have the same position relative to elements from other chains and can be considered the same. This rule is intended to reduce the number of redundant possibilities. This step partially satisfies A_1 and A_5 and does not violate any of the other assumptions. It does not have an effect on example sentence (12) but is useful in a case like that of sentence (13):

(13) Everyone loves someone.

Sentence (13) is shown in Figure 11 after applying Step 3A. In Figure 11, *someone(1)* and *someone(2)* form one chain with *everyone(1)* the only remaining element in its chain. *Someone(1)* and *someone(2)* are both to the right of *everyone(1)* and do not

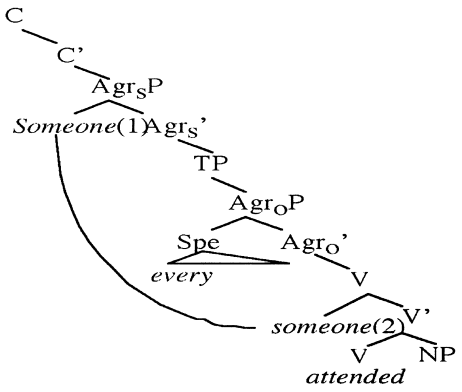

FIGURE 10. The element *every seminar(2)* removed in sentence (12).

have a link from another chain between them so we can delete one of the two *someone*s to reduce the number of redundant possibilities.

Step 3C satisfies assumption A_1. In this step we obtain the two possible interpretations for sentence (12) that are shown in Figure 12.

The algorithm for Step 3C is given below.

Algorithm for step 3C

```
For each element in chain 1 Do
   For each element in chain 2 Do
      ...
      For each element in chain N Do
         Save selected element from each chain in a list.
      End For
      ...
   End For
End For
For each list of elements Do
   Copy the derivation
```

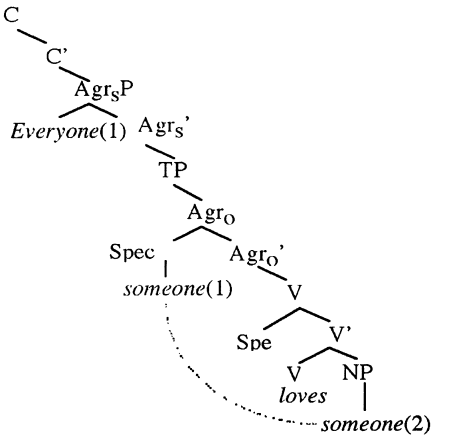

FIGURE 11. The element *everyone(2)* removed in sentence (13).

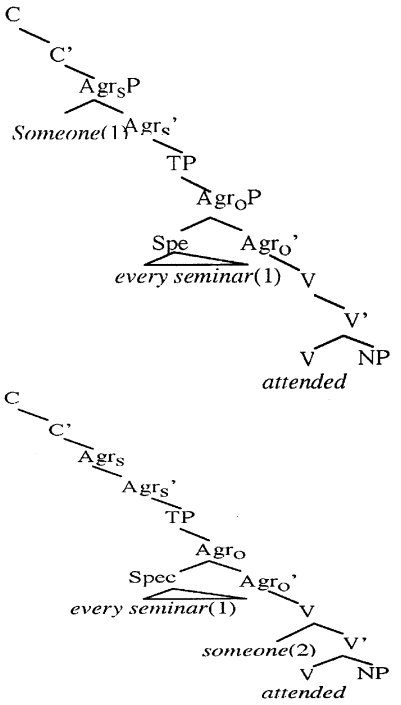

FIGURE 12. The two possibilities for scope: *someone* with wide scope or *every seminar* with wide scope.

```
        Traverse the derivation removing all the elements
            that are not within in the list.
    End
```

For each chain there is a FOR loop that iterates through each element. The loops are nested so that at each iteration through the innermost loop there will be a different combination of one element from each chain. Each combination of elements is saved. For each combination a copy of the derivation is made and all the elements that are not in the combination are removed from the derivation.

4.4. Handling Raising Verbs

The same simple approach due to Hornstein to handle quantifier scope can be used to handle several other linguistic phenomena. In this section and the following, we will discuss two such phenomena and show how we adapt the LF processor to handle each case.

The first phenomenon we discuss concerns *raising verbs*. In English, verbs such as *seem* and *intend* are called raising verbs in Principles and Parameters grammars (Huddleston 1993, p. 226) because there are situations in which they force the subject to move from an embedded sentence to the outer or containing or matrix sentence. Huddleston (1993, Chap. 5) puts verbs into two classes: (1) those in the *seem* class are: *begin, appear, happen, fail, need, tend, finish, stop*, and *continue*; those in the *intend* class are *desire, hate, like, want, believe, claim, know, report*, and *think*. Verbs belonging to these two classes are called *raising verbs* [Hudon, p. 226] in principles and parameters grammars.

The approach outlined by Hornstein is able to handle, in a straightforward manner, many issues that complicate matters with regard to the computational processing of sentences containing raising verbs. We can handle the following and similar sentences.

(14) Someone seemed to attend every class.
(15a) Someone persuaded John to attend every class.
(15b) Someone hoped to recite every poem.
(15c) Someone believes John to be attending every class.
(16a) Someone expected every Democrat to win.
(16b) Someone expected every Democrat would win.

Sentence (14) is ambiguous to many with regard to quantifier scoping. Sentences (15) are unambiguous. Sentence (16a) is ambiguous to many, whereas sentence (16b) is unambiguous.

For sentence (14), after the NPs have moved overtly and covertly, we have the following syntactic structure at LF:

[Agr_sP Someone$_1$ seemed [Agr_sP someone$_2$ to [Agr_oP (every class)$_1$ [VP someone$_3$ attended (every class)$_2$]]]].

The term *every class*$_2$ is a definite NP and, following Hornstein's assumption A_3, it is deleted from inside the VP shell. Now, we are left with a chain containing three instances of *someone*, and a chain containing only one instance of *every class*. All but one instance of *someone* must be deleted following assumption A_1. If we keep either *someone*$_1$ or *someone*$_2$ by deleting all the others, *someone* takes wider scope over *every class* following assumption A_2. If we keep *someone*$_3$ and delete the others, we get the other meaning where *every class* has wider scope. These two give us the ambiguity in the sentence.

In the case of sentences (15a), (15b), and (15c), *someone* starts from the internal position of the matrix VP (i.e., the VP that contains the embedded sentence). In each case, *every class* starts within the internal position of the embedded VP. Whichever elements of the resulting chains we delete, it is impossible to get the indefinite NP *someone* of the matrix sentence inside the scope of the definite NP *every class* of the embedded sentence. This rules out ambiguity. In older versions of principles and parameters grammars, one had to postulate other means for solving this problem.

The fact that sentence (16a) is ambiguous and sentence (16b) is not can also be shown in a similar fashion.

4.5. Handling Possessives

We now discuss how we can extend our algorithm to handle simple possessives as in the sentence:

(17) Everybody beat his donkey.

Here, there is a possessive *his* used as a determiner in the noun phrase *his donkey*. In such a situation, *his* is considered a definite quantifier. Following Hornstein, we can simply delete *his donkey* from inside the VP shell.

In sentence 17, the structure at LF is the following before deletion:

[Agr_sP every man [TP [Agr_oP his donkey [VP every man [V' beat his donkey]]]]].

Now, we need to delete the definite NPs from inside the VP shell. Here, both *every man* and *his donkey* are definite NPs.

So, after deletion, we have the following, with deleted NPs in parentheses:

[Agr_sP every man [TP [Agr_oP his donkey [VP (every man) [V′ beat (his donkey)]]]]].

Now we have *every man* taking scope over *his donkey*. In this resultant structure, *every man* c-commands *his donkey*. Using Binding Theory we say that when the antecedent c-commands the pronoun, binding is permitted. Thus, we can bind *every man* with *his donkey*. Thus we have one donkey for each man if we do the binding.

Binding Theory is the module of GB and MP grammar that regulates the referential properties of NPs (Haegeman 1994). The Binding Theory, which was enunciated in the context of the Government and Binding Theory, applies in the context of the MP also. It must be applied at the LF—the only phrase structure form available in MP.

But, the statement of Binding Theory says "binding is permitted," which means that although binding is permitted, it is not necessary. If we bind, we get one meaning, and if we do not bind we get the other meaning. This gives us the two meanings of the sentence.

Binding resolves the ambiguity in sentences like (17) and it is not a matter for scoping. However, we do not believe we should add this piece of Binding Theory to our algorithm unless we deal with the general binding problem. If we were to add it, a reader is likely to see this small binding piece we have added to deal with a particular problem and say "This seems to be a convenient solution to this problem but where is your general binding solution?" We leave a complete implementation of the Binding Theory to future research.

4.6. Handling Other Syntactic Phenomena

Many other syntactic phenomena that needed specialized techniques and principles in older principles and parameters grammars can be handled using the approach espoused by Hornstein. Some examples of such phenomena are Empty Category Principle (ECP), weak crossover, superiority, and polarity effects, and many instances of interactions between binding and relative quantifier scope in addition to the simple case of such interaction we saw earlier. Our LF processor can be easily adapted to handle all of these phenomena. Since the linguistic concepts are difficult for the general AI reader, we do not discuss them. However, we must emphasize that because Hornstein's approach obviates the need for special operations such as quantifier raising, the computational problems in implementation of an LF processor are simplified.

4.7. Comparison with Stabler's Approach to Quantifiers

For quantifier phrases, Stabler (1978, and forthcoming) adopts a simple version of the theory of quantifiers proposed by Szabolcsi (1997) and Beghelli and Stowell (1997). He distinguishes the following four categories of quantifiers:

1. Negative determiners, such as *no* as in *No boy showed up.*
2. Distributive or universal determiners such as *each* and *every*.
3. Group denoting determiners such as *the, some, a, one, three*.
4. Counting determiners such as *few* and *fewer than five*.

Furthermore, following Beghelli and Stowell, Stabler assumes new special functional categories, named `ref`, `dist`, and `foc` (or, `share`), to handle quantifiers. They provide specifier positions that distinguish among quantifiers. Quantifiers of a certain type move to the specifier of a certain predesignated functional category to have their features checked. In particular,

1. Negative quantifiers are interpreted in the specifier of the functional category *neg*.
2. Distributive and universal quantifiers are interpreted in the specifier of the functional category *dist*.
3. Group denoting quantifiers are interpreted in the specifiers of the functional categories *ref*, *foc* or in their case positions.
4. Counting determiners are interpreted in their case positions.

For purposes of derivation, Stabler allows the two basic movements that we discussed earlier: generalized transformation and singulary transformation, although he uses them with different names for them. However, in a somewhat nonstandard fashion (linguistically speaking), he divides singulary transformation (where a part of a structure moves from a lower level and attaches to the top) into three separate movements: only the phonetic features move, only the semantic features of a lexical item moves, all the features (i.e., both semantic and phonetic features) move. Additionally, a quantifier (e.g., *some*) of a functional category (e.g., *ref*) may have several different lexical entries with different sets of features. The combination of four different kinds of movements and several lexical items for some of the quantifiers and functional objects allows different possibilities for the resultant structure and associated ambiguities in interpretation of sentences with multiple quantifiers. It is a clever approach, but we found the alternative feature structure sets for quantifiers and functional objects difficult to motivate. Also, because of the several new functional heads that are introduced, the derivation process becomes lengthier and more complex.

Although Stabler has detailed outlines regarding how things are supposed to work, we have not seen any description of a working program. Also, the sentences he handles are much simpler those we have covered in our work.

4.8. Determining a Logical Representation Compositionally

A logical form representation can be determined compositionally using a semantic lexicon and semantic rules associated with each grammar rule. We use the Montague-semantics-based approach outlined in Dowty (1979), Dowty et al. (1992), and Chierchia (1993) to obtain the semantic representation in first-order predicate logic. For detailed discussion, the reader is referred to Williams (1997).

5. DESCRIPTION OF THE PROGRAM

The program is written in Common Lisp using the GCL (GNU Common Lisp) interpreter that runs on Unix. It is a pipeline where the outputs of the previous step become the inputs to the next step. For the first step in the pipeline we use Allen's bottom-up parser (Allen (1995), Chaps. 3–5) with the grammar described previously. The second step is to build an approximate derivation structure at Spell-Out for each of the syntactic parses. This step creates the empty nodes, labels all the argument positions, and builds chains following our chain-building algorithm. The third step in the pipeline is to build chains for covert movements. Finally, we apply our Step 3 algorithms to produce derivations that have the correct quantifier scoping and can be used to determine a logical representation compositionally. Here, we show only the simplest example. Details of other examples can be found in Williams (1997).

5.1. Details for *John loves Mary*

We obtain the syntactic parse by calling the routine `get-parse-trees`. This will result in a list of parse trees, the first of which is shown next:

```
(AGRS2 7 ((1 N24) (2 AGRS129)) (N2 25 ((1 N0))
   (N NIL ((LEX JOHN))))
   (AGRS1 10 ((1 T228))
      (T2 11 ((1 AGRO227))
         (AGRO2 14 ((1 AGRO126))
            (AGRO1 16 ((1 V2TRAN25))
               (V2TRAN 18 ((1 V1TRAN24))
                  (V1TRAN 22
                  ((1 VTRAN1) (2 N223))
                  (VTRAN NIL ((LEX LOVES)))
                  (N2 25 ((1 N3))
                  (N NIL ((LEX MARY)))))))))))
```

The parse is one list. The first element is the category. The second element is the rule number. The third element is a list showing the instances of the rules that are the direct descendants. Finally the fourth and fifth elements are the descendants, which are lists defined the same way as the parent.

The next command performs node labeling, empty node creation, and chain building to give us an approximate derivation structure at the point of Spell-Out. Running the routine called `get-parses-with-chains` results in:

```
(AGRS2 7 NIL
   (N2 25 ((CHAIN 1) (LEX JOHN) (LABEL AH) (POS 10))
      (N NIL))
   (AGRS1 10 NIL
      (T2 11 NIL
         (AGRO2 14 NIL
            (AGRO1 16 NIL
               (V2TRAN 18 NIL
                  (EMPTY NIL
                   ((CHAIN 1) (LEX JOHN)
                    (LABEL AF) (POS 16)))
                  (V1TRAN 22 NIL
                   (VTRAN NIL ((LEX LOVES)))
                   (N2 25
                    ((CHAIN 2) (LEX MARY)
                     (LABEL AH) (POS 17))
                    (N NIL)))))))))
```

As before, the derivation structure is shown in one list. The first element is the category and the second is the rule number. Here, however, the third element is NIL or a feature list for argument positions. One of the feature lists is:

```
((CHAIN 1) (LEX JOHN) (LABEL AH) (POS 10)).
```

This feature list shows that the argument is part of chain 1, has the lexeme *John*, is labeled the head of an A-chain, and is uniquely identified as position 10. As in the syntactic parse, the fourth and fifth elements, if they exist, are the descendants which are themselves lists the same format.

If we look at the derivation closer we can see another argument with the lexeme *John* that is labeled the foot of an A-chain and identified as position 16. This argument is also part of chain 1 and is the foot of the chain. This is the base position for the chain. The category is EMPTY meaning that it was an empty position that was created and filled with the chain-building rules.

The argument with lexeme *Mary* is the only member of its chain (chain 2), since we are just at the point of Spell-Out and have not yet shown the movements for checking weak features. Invoking a subroutine called `CWF` (which stands for *check weak features*) gives us the derivation after our step 2, where covert chains are built:

```
((AGRS2 7 NIL
    (N2 25 ((CHAIN 1) (LEX JOHN) (LABEL AH) (POS 10))
       (N NIL))
    (AGRS1 10 NIL
       (T2 11 NIL
          (AGRO2 14 NIL
             (N2 25
                ((CHAIN 2) (LEX MARY) (LABEL AH)
                (POS 117))
                (N NIL) NIL)
             (AGRO1 16 NIL
                    (V2TRAN 18 NIL
                       (EMPTY NIL
                        ((CHAIN 1) (LEX JOHN)
                         (LABEL AF) (POS 16)))
                       (V1TRAN 22 NIL
                        (VTRAN NIL ((LEX LOVES)))
                        (N2 25
                         ((CHAIN 2) (LEX MARY)
                          (LABEL AH) (POS 17))
                         (N NIL)))))))))
```

We can see a new element with lexeme *Mary* that is in the [Spec, Agr_o] position, the category `N2` as the first child of `AGRO2`. Finally, if we execute our Step 3 algorithms we obtain the final parse, which is not much different.

This derivation selected *John* and *Mary* both in their head positions; the other combinations of *John* and *Mary* in their other chain positions are generated but are not shown here.

6. COMPARISON WITH OTHER PRINCIPLES-BASED PARSERS

We briefly compare our effort at parser writing with those reported in various published papers on principles-based parsing. As we have noted earlier, as far as we know, ours is the first attempt at building a substantial program that obtains the parse of a sentence conforming to the MP. Stabler's papers discuss a computational approach to the derivation of sentences in the MP and handling of quantifier scope issues Stabler 1997, an forthcoming. However, we have not seen any discussion on an implemented parser and post-parse or LF processor. Hence, our discussion here is essentially concerned with parsers based on an earlier version of Principles and Parameters grammar, the Government and Binding Theory (GB Theory).

Surely, there are examples of efforts at writing very substantial grammars for various natural languages, such as the XTAG project at the University of Pennsylvania (XTAG 1998). However, such efforts are not interested at all in basing their work on any cognitive theory or principles, even partially.[1] In general, the coverage of principles-based parsers is small because such parsers are still products of exploratory research.

[1] Private communication with Aravind Joshi, the principal behind the XTAG project.

Our processor was built to cover a set of sentences that have been used by Merlo in his paper describing a parser that was based on the GB Theory (Merlo 1995). We have added another set of sentences to illustrate how the scope of quantifiers and the interaction between quantifier use and possessives can be handled. We believe that it is a sufficiently large syntactic coverage for a program meant primarily to show how a parser and an LF processor based on the MP can be written.

Two of the related main computational difficulties in building parsers that are built to work with principles and parameters only are *overgeneration* and *slow parsing* (Berwick, Abney, and Tenny 1992, p. 8). Because exclusive binary branching makes the syntax trees very deep, the number of functional entries can become large and there are many lexical entries (including several for each word), with the result that the principles vaguely talk about economy without being explicit. Hence, as a compromise, we have obtained rules based on a study of derivations of a large number of sentences.

Of course, it will be a great idea if we can derive the rules from the basic principles of the MP. We have already started working on aspects of this problem. Our ongoing research, not reported in this paper, attempts to give precision to the economy conditions discussed in the MP. We have written a program that takes various lexical entries and simulates the movements that the entries undergo during the process of derivation of a sentence, and the optimality-driven causes for such movements. This will help us understand the nature of the optimality or the economy conditions in precise terms. Other researchers are working on expounding them in theoretical linguistic terms as in Kitahara (1997) and Collins (1997). It will be a significant triumph if the rules can be obtained automatically, but it is a huge challenge left for future research.

Berwick and collegues write that all the authors in their book (Berwick et al. 1992, p. 32) who wrote about building parsers based on GB, viz., Abney (1992); Correa (1992); Dorr (1992); Fong (1992); Epstein (1992); Kashket (1992); and Johnson (1992) made compromises as far as the modularity of the various theories and principles (there are about twenty of them in GB). They put together several modules (theories and/or principles) to form super modules. They did this to reduce the problems of overgeneration and slow processing. Berwick commented that they did not lump all the principles together because that would lead us back to rules again. However, we do not see even this as wrong-headed; this was precisely the approach taken by Merlo (1995). There are many standard, traditional, and efficient techniques for handling rules in compiler design. People who work seriously with the design and implementation of compilers will consider it unreasonable, with proper justification, if a traditional and proven technique is not used. However, not everything is lost here. In the ideal case, we should be able to obtain these rules programmatically based on the theories and principles. The problem here is that the principles are vaguely worded. Even if they were not, an off-line a priori computation of the rules will be the way to go. Our program does not do so, and we leave it for future research.

Adapting the comments in Berwick et al. to our effort, we can say that what we have done looks more like a conventional context-free, rule-based system. However, the synthesized rules embody the principles of MP, and later principles like Binding Theory still need to be applied at LF. The process of synthesizing several principles into a super principle or into rules can be considered "compiling." In the parsers reported in Berwick et al., the authors took two approaches to this compile-time/run-time tradeoff. Some, like Dorr (1992), Epstein (1992), Fong (1992), and Kashket (1992) did most of the work by hand; others like Johnson (1992) have taken a more automatic route. We have used the by-hand approach, but would like to automate or semiautomate the process in the near future.

7. CONCLUSIONS

There are several other Principles- and Parameters-Based parsers, such as those of Dorr (1992) and Lin (1993) that are based on the GB theory that predates the Minimalist Program. GB principles constrain X-bar structures. The principles are applied in isolation on an X-bar structure to determine its grammaticality. Dorr essentially builds all possible X-bar structures for a sentence and then applies the principles as each structure is built, filtering out the illicit structures. Lin's parser, which is network based and is quite efficient, is also based on GB theory but Lin applies the GB "principles to descriptions of X-bar structures rather than the structures themselves."

The primary limitation of our system is that it only works for sentences with the same syntactic structure as our example sentences. However, due to the Minimalist Program being a relatively new theory there is not yet work outlining detailed examples for an extensive number of sentences in an uncontroversial manner, such as there is for Government and Binding Theory (Haegeman 1994; Lasnik and Uriagereka 1988; van Riemsdijk and Williams 1986). Williams (1997) obtains a significant set of derivations for sentences worked out to the precision that is needed for writing computer programs. These derivations are glossed over in this paper in Section 2. Since Principles and Parameters approaches are intended to describe all natural languages, a natural extension to our work would be to incorporate coverage for other languages. One of the authors is working on applying the principles of the MP to some of the easternmost Indo-European languages such as Assamese and Bengali.

The efficiency of the algorithms has not been our primary concern. In order to improve program performance one may be able to incorporate the algorithms within the parsing mechanism instead of explicitly creating and traversing the derivation trees.

Lin's (1993, 1994) principles-based parser applies principles to descriptions of structures instead of the structures themselves. A structure description could be built for the Minimalist Program by basing the network on the clause structure and adding more constraints following the principles of *Shortest Move, Procrastinate*, and *Greed*. The algorithms discussed in Sections 3 and 4 could possibly be built into the structure description itself. For example, the network may look like the one given in Figure 13 in which nodes are positions where overt elements are found. All the nodes are linked to a central node which collects message items until it determines that a clause is completed. The message item for a complete clause is then sent to the S node. Message items would be as described in Lin (1994). An item is a triplet that describes a structure ⟨surface-string, attribute-values, sources⟩ where surface-string denotes the contiguous words in the sentence, attribute-values specify syntactic features, and sources describe the immediate substructures. The S node would collect completed clause structures until it determined

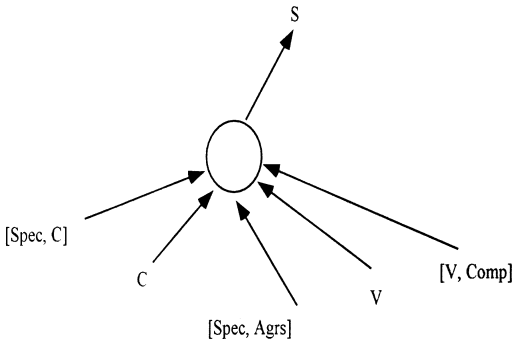

FIGURE 13. A possible network for message-based parsing with the MP.

the sentence was complete. Since the Minimalist Program has explained linguistic theory in a more uniform and simpler way than Government & Binding theory, this parser may be simpler with coverage equivalent to that of Lin (1993, 1994).

REFERENCES

ABNEY. S. P. 1992. Parsing by chunks. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 257–278.

ALLEN, J. 1995. Natural Language Understanding, 2nd ed. Benjamin Cummings Publishing, Menlo Park, CA.

AOUN, J., and A. LI. 1993. The Syntax of Scope. MIT Press, Cambridge, MA.

BEGHELLI, F., and T. STOWELL. 1997. Distributivity and negation: The syntax of *each* and *every*. *In* Ways of Scope Taking. Kluwer Academic Publishers, Dordrecht.

BERWICK, R. C., S. P. ABNEY, and C. TENNY, Ed. 1992. Principle-Based Parsing: Computation and Psycholinguistics. Kluwer Academic Publishers, Dordrecht.

CHIERCHIA, G., and S. MCCONNELL-GINET. 1993. Meaning and Grammar: An Introduction to Semantics. MIT Press, Cambridge, MA.

CHOMSKY, N. 1981. Lectures on Government and Binding. Foris, Dordrecht.

CHOMSKY, N. 1993. Language and Thought. Moyer Bell, Wakefield, RI.

CHOMSKY, N. 1994. Bare phrase structure. MIT Occasional Papers in Linguistics 5, Department of Linguistics and Philosophy, MIT, Cambridge, MA.

CHOMSKY, N. 1995. The Minimalist Program. MIT Press, Cambridge, MA.

COLLINS, C. 1997. Local Economy. MIT Press, Cambridge, MA.

CORREA, N. 1992. Empty categories, chain binding, and parsing. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 83–122.

CULICOVER, P. W. 1997. Principles and Parameters: An Introduction to Syntactic Theory. Oxford University Press, UK.

DIESING, M. 1992. Indefinites. MIT Press, Cambridge, MA.

DORR, B. J. 1992. Principle-based parsing for machine translation. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 153–184.

DOWTY, D. R. 1979. Word Meaning and Montague Grammar. R. Kluwer Academic Publishers, Dordrecht.

DOWTY, D. R., R. E. WALL, and S. PETERS. 1992. Introduction to Montague Semantics. Kluwer Academic Publishers, Dordrecht (reprint of 1981 edition, corrections).

EPSTEIN, S. 1992. Principle-based interpretation of natural language quantifiers. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 185–198.

FONG, S. 1992. The computational implementation of principle-based parsers. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 65–82.

HAEGEMAN, M. V. L. 1994. Introduction to Government and Binding Theory. Oxford Univerity Press, UK.

HORNSTEIN, N. 1995. Logical Form, From GB to Minimalism. Blackwell Publishers.

HUDDLESTON, R. 1993. Introduction to the Grammer of English. Cambridge University Press, Cambridge, England (reprint of 1984 edition).

JOHNSON, M. 1992. Deductive parsing: The use of knowledge of language. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 30–64.

KASHKET, M. B. 1992. Parsing Walpiri—A free word order language. *In* Principle-Based Parsing: Computation and Psycholinguistics. *Edited by* R. C. Berwick, S. P. Abney, and C. Tenny. Kluwer Academic Publishers, Dordrecht, pp. 123–152.

KITAHARA, H. 1997. Elementary Operations and Optimal Derivations. MIT Press, Cambridge, MA.

LASNIK, H. 1999. On feature strength: Three minimalist approaches to overt movement. Linguistic Inquiry, **30**(2):197–218.

LASNIK, H., and JUAN URIAGEREKA. 1988. A course in GB Syntax: Lectures on Binding and Empty Categories. MIT Press, Cambridge, MA.

LIN, D. 1993. Principle-based parsing without overgeneration. *In* Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, Columbus, Ohio.

LIN, D. 1994. PRINCIPAR—An efficient, board-coverage, principle-based parser. *In* Proceedings of COLING: the International Conference on Computational Linguistics.

MARANTZ, A. 1995. Government and Binding Theory and the Minimalist Program, Blackwell, Oxford, UK.

MERLO, P. 1995. Modularity and information content classes in principle-based parsing. Computational Linguistics, **21**(4):515–541.

MOHANAN, T. 1994. Argument Structure in Hindi. CSLI Publications, Stanford University, Stanford, CA.

STABLER, E. P. 1997. Computing quantifier scope. *In*. Ways of Scope Taking. *Edited by* Kluwer Academic Publishers, Dordrecht, pp. 155–192.

STABLER, E. P. Forthcoming. Parsing and generation for grammars with movement. http://www.humnet.ucla.edu/humnet/linguistics/people/stabler/epspub.htm.

STROIK, T. 1996. Minimalism, Scope and VP Structure. SAGE Publications, Thousand Oaks, CA.

SZABOLCSI, A. 1997. Strategies for scope taking. *In* Ways of Scope Taking. *Edited by* Kluwer Academic Publishers, Dordrecht, pp. 109–154.

THE XTAG GROUP. 1998. The XTAG project, http://www.cis.upenn.edu/x̃tag/home.html. Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

VAN RIEMSDIJK, H., and E. WILLIAMS. 1986. Introduction to the Theory of Grammar. MIT Press, Cambridge, MA.

WILLIAMS, J. S. 1997. Natural language processing and the minimalist program. MS thesis, Department of Computer Science, University of Colorado at Colorado Springs.

ZWART, C. J.-W. 1994. Introduction. Gröningen Arbeiten zur germanistischen Linguistik, **37**:1–17.