Implementation of Pronoun Readings in English: A Categorial Grammar Approach

Yong-hun Lee
(University of Illinois at Urbana-Champaign)

Lee, Yong-hun. 2001. **Implementation of Pronoun Readings in English: A Categorial Grammar Approach**. *Korean Journal of English Language and Linguistics* 1-4, 609-627. Pronouns are frequently used in English, and their resolution is important to capture meaning of sentences. This paper provides a computational implementation for pronoun readings in English, based on Chierchia's (1988) Binding Theory in Categorial Grammar. A CCG-like system is newly devised for implementing his ideas, where syntactic phenomena are represented by the functor-argument relations of categories. This relation triggers resolution algorithms, and reflexives and pronominals are resolved succinctly. In sum, this paper gives an efficient resolution algorithm for English pronouns within Categorial Grammar.

1. Introduction

Pronouns are frequently used in English, and their resolution is crucial to understand the meaning of sentences. It is also important to natural language processing and machine translation.

In order to capture the relations between pronouns and their antecedents, many scholars have developed so called Binding Theories. There are roughly two types of Binding Theories. One of them uses tree-theoretic notions and heavily depends on structural configurations. Chomskyan traditions such as GB framework belong to this type. The other explains syntactic dependencies by functor-argument relations. These kinds of analyses start from Bach and Partee (1980), and Binding Theory in Categorial Grammar is an example.

The goal of this paper is to provide an efficient computational

algorithm for pronoun resolution in English. As a theoretical base, Binding Theory in Categorial Grammar is adopted, which was described in Chierchia (1988). But, in its implementation, rather than follow his original formalism, I will combine his basic ideas with Seedman's (1996, 2000) Combinatory Categorial Grammar, which will be called a CCG-like system.

2. Binding Theory in Categorial Grammar: Chierchia (1988)

Categorial Grammar was first introduced by Ajdukiewicz (1935) and later modified and advanced by Bar-Hillel, Curry, and Lambek. In this framework, we have two basic categories *n* and *s*, and other categories come from the combinations of these two categories. All the syntactic phenomena are described and analyzed by the functor-argument relations of categories.

Steedman (1996, 2000) extended previous studies in Categorial Grammar and developed Combinatorial Categorial Grammar (CCG). The most important characteristic of his system is that predicate-argument relations are projected by the combinatory rules of syntax, and other operations are based on these relations (Steedman 2000:38). Two basic combinatory rules are *functional application* and *functional composition*, which are delineated in (1).

(1) Two Basic Combinatory Rules

a. Functional Application

(i)	X / Y	Y	⇒	X
(ii)	Y	X \ Y	⇒	X

b. Functional Composition

(i)	X / Y	Y / Z	⇒	X / Z
(ii)	X / Y	Y \ Z	⇒	X \ Z
(iii)	Y \ Z	X \ Y	⇒	X \ Z
(iv)	Y / Z	X \ Z	⇒	X / Z

Chierchia used Categorial Grammar to account for Binding phenomena in English, and he described syntactic constraints of reflexives and pronominals as follows.

(2) Binding Theory in Categorial Grammar (Chierchia 1988:134)
   a. A reflexive must be bound to an F-commanding argument in its minimal NP or S domain.
   b. A non-reflexive pronoun must not be co-indexed with anything in its minimal NP or S domain.
   where F-command is simply c-command at function-argument structure.[1)]

Agreement in *number* and *gender* must hold between pronouns and their antecedents, in order to pronouns can refer to their antecedents. The constraint for checking agreement is stated in (3a). FT(n) in (3b) has three information: n is the index of the given NP, *gndr* is gender, and *nmbr* is number.

(3) Agreement between Antecedent and Pronouns (Chierchia 1988:132)
   a. FT(n) $\approx$ FT(m): The features associated with n are non-distinct from those associated with m.

---

[1]According to Chierchia, F-command has different prediction for the sentences in (i) and (ii). It rules in (i), but rules out (ii) (Chierchia 1988:135).

(i) Mary showed the men each other.
(ii) *Mary showed each other the men.

He claimed that Binding Theory in GB says cannot differentiate two sentences. Both sentences are grammatical because *each other* and *the men* c-command each other. Note that, however, two sentences can be explained if we assume Larson's (1988) analyses.

b. $FT(n) = \begin{bmatrix} n \\ \text{gndr} \\ \text{nmbr} \end{bmatrix}$

For example, FTs of three different NPs *John*, *himself*, and *her* can be stated as follows[2].

(4) John_1 himself_2 her_3

$FT(1) = \begin{bmatrix} 1 \\ \text{male} \\ 3 \end{bmatrix}$ $FT(2) = \begin{bmatrix} 2 \\ \text{male} \\ 3 \end{bmatrix}$ $FT(3) = \begin{bmatrix} 3 \\ \text{female} \\ 3 \end{bmatrix}$

Chierchia introduced resolution algorithms for pronouns based on the combinatory rules of categories, and it can be enumerated in (5). For each rule in (5a)-(5c), two constituents can be combined iff all the four conditions are satisfied. If not, we cannot combine the constituents.

(5) Chierchia's Algorithms (1988:138-9)

a. $\underset{0}{\text{TV}} + \underset{1}{\text{NP}} \Rightarrow \underset{2}{\text{IV}}$ (here and throughout integers will be used as names for the categories mentioned in the rules)

conditions:	(i)	$LPS(0) \cap LPS(1) = \emptyset$[3]	non-coreference
	(ii)	$SLASH(2) \cap (LPS(1) \cup LPS(2)) = \emptyset$[4]	crossover
	(iii)	$SLASH(2) = SLASH(0) \cup SLASH(1)$	slash-percolation
	(iv)	$LPS(2) = LPS(0) \cup LPS(1)$	LPS-percolation

[2]Note that *John*, *himself*, and *her* stand for an R-expression, a reflexive, and a pronominal respectively.

[3]LPS (Local Pronoun Store) stores indices for pronouns and their antecedents. See (8d).

[4]SLASH is a storage where missing elements in the argument structure are stored. See (8e).

b. $S/NP_n + NP_n \Rightarrow S$
$\quad 0 \quad 1 \quad 2$

conditions:			
	(i)	LPS(2)=Ø[5]	A-opacity boundary
	(ii)	SLASH(2) ∩ (LPS(1) ∪ LPS(2))=Ø	crossover
	(iii)	SLASH(2)=SLASH(0) ∪ SLASH(1)	slash-percolation
	(iv)	$n \notin$ LPS(0) ∪ LPS(1)reflexive	
		$\quad +refl$	

c. $IV/IV + IV \Rightarrow IV$
$\quad 0 \quad 1 \quad 2$

conditions:			
	(i)	LPS(2)=LPS(0)	A-opacity boundary
	(ii)	SLASH(2) ∩ (LPS(1) ∪ LPS(2))=Ø	crossover
	(iii)	SLASH(2)=SLASH(0) ∪ SLASH(1)	slash-percolation
	(iv)	$n \notin$ LPS(1)reflexive	
		$\quad +refl$	

d. Reflexives

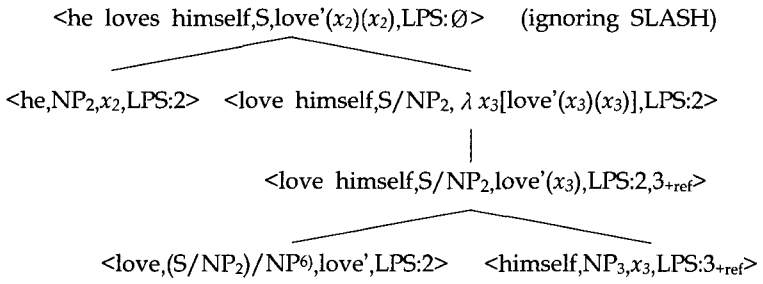
(i) $\underset{n[+refl] \in LPS}{A} \Rightarrow \underset{n \notin LPS}{A}$

(ii) conditions: (a) A = IV, TV (b) $FT(A) \approx FT(n)$

(iii) translation: $\lambda x_n\ [A'(x_n)]$

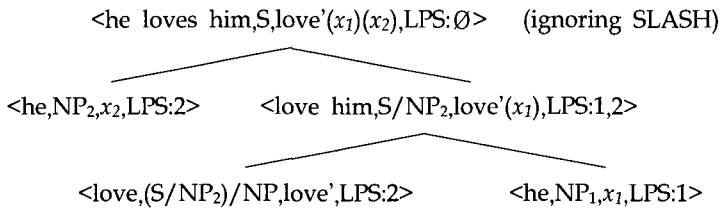
According to these algorithms, two pronouns *himself* and *him* are resolved as in (6) and (7). Here note that the translations of two sentences are different, i.e., $love'(x_2)(x_2)$ vs. $love'(x_1)(x_2)$.

(6) He loves himself.



[5]As discussed in Lee (in prep), this condition is English-specific. It must be modified to deal with long-distance reflexives in Korean.

(7) He loves him.



Because *himself* is a reflexive pronoun, (5d) is applied in the analysis in (6), and changes *love'*(x_3) into λx_3[*love'*(x_3)(x_3)]. This operation does not occur in (7) because *him* is a pronominal, which results in different translation between reflexives and pronominals.

3. Implementation of Pronoun Readings

3.1. Basic Ideas of a CCG-like System

There are two ways of implementing Chierchia's ideas. The first one is to make use of translations, which were generated by the combination of two categories. The second is to utilize the indexes of NPs, which are stored in LPS and SLASH. In its actual computational algorithms, the former method is more difficult to implement on a computer than the latter, because the translations themselves are represented by predicate logic. Accordingly, this paper chooses the second option, and all the pronouns are resolved by manipulating the indexes of NPs.

The system that this paper wants to develop is a CCG-like system, which is a little modified from Steedman's CCG. This modification is necessary to implement Chierchia's algorithms more easily and more effectively. This system is similar to Steedman's system in that combinatory rules triggers other operations, especially pronoun resolution algorithms. It is

[6]Here 2 in (S/NP$_2$)/NP means the index of subject NP must be 2. It is a tool for subject-predicate agreement.

different from Steedman's because it has five attributes to describe syntactic dependencies.

The five attributes and their roles in this system are explained in (8).

(8) Five Attributes
   a. PHON: concatenates a word to a stream of words.
   b. CAT: has categorial information, such as S, NP, S/NP, and so on.
   c. AGR
      (i) agreement feature
      (ii) index, type[7], gender, and number
   d. LPS (Local Pronoun Store)
      (i) something like a Cooper-storage
      (ii) has indices for pronouns and their antecedents.
   e. SLASH
      (i) similar to that of HPSG, except that it deals with pronouns
      (ii) necessary to deal with crossover phenomena

For NPs, we have three different types of indexes. +refl and +pron are attached to the indexes of reflexives and pronominals respectively, but nothing is added to those for proper nouns. The index types of NPs in this system are summarized in (9).

(9) Three Index Types of NPs
   a. n : proper nouns
   b. n_{+refl} : reflexives
   c. n_{+pron} : pronominals

[7]Here, *type* is one of the index types in (9).

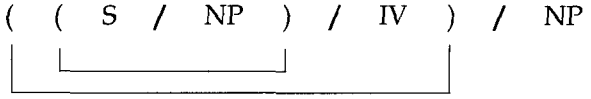
Out of two combinatorial rules in (1), this paper will use the *functional application* (1aii) only. Functional applications on CAT values decide grammaticality of the input sentences. The functor-argument relations are determined based on the CAT value of each category. Operations on CAT value trigger those on LPS and SLASH, and pronouns are resolved by the interactions of LPS and SLASH values.

3.2. Algorithms for Categories

Before we start to develop algorithms for pronoun resolution in English, we need two more mechanisms for handling categories. One is recognition or separation of categories, and the other is implementation of functional application.

A category has a form A or A/B. When we have an atomic category NP or S[8], what we have to do is just returning the categories. When we have a complex category such as ((S/NP)/IV)/NP, we need to separate this one into two categories, (S/NP)/IV and NP. The important clues are the nested structure of parentheses and a slash (/) between two categories. That is, our example category can be analyzed as in (10).

(10) Nested Structure of Parentheses



In its computational implementation, a stack meets our purpose. A stack is a LIFO-style storage, where LIFO means Last In First Out. We have two operations PUSH and POP for the stack. PUSH stores something into the stack, whereas POP removes the top-most content from the stack. In its computational implementation, we PUSH a left parenthesis ('(')

[8]The atomic categories of this system are S and NP, instead of *n* and *s*.

into the stack when we meet it, and POP a left parenthesis when we get a right parenthesis (')'). But we don't need to use a real stack. We can use a virtual stack instead, because the contents of the stack are identical. The concrete algorithm is illustrated in (11).

(11) Algorithm for Separating a Category

```
function string separate_category(string s)
var
    string A, B;
begin
    if there is no / in s then return s;
    else if there is one / and no ( in s then
        begin
            put the string before / into A;
            put the string after / into B;
            return(A, B);
        end
    else
        begin
            while read through s do
                begin
                    if there is a ( then PUSH('(');
                    else if there is a ) then POP('(');
                end;
            if the stack is empty then
                put the string up to now into A;
            if the next string is not / then return error;
            while read through s do
                begin
                    if there is a ( then PUSH('(');
                    else if there is a ) then POP('(');
                end;
            if the stack is empty then
                put the string up to now into B;
            if the stack is empty then return(A, B)
            else return error;
        end;
end;
```

The algorithm for functional application is as in (12). In *func_application*(s_1, s_2), s_1 is a functor and s_2 is an argument. This function checks if s_2 is the argument of s_1. If it is, the function returns the result of functional application. If not, it returns an error message.

(12) Algorithm for Functional Application

```
function string func_application (string s1, string s2)
var
    string A, B;
begin
    A, B = separate_category(s1);
    if s2 = B then return A else return error;
end;
```

When s_1 is an argument of s_2, the position of s_1 and s_2 is interchanged outside of this function and functional application is tried again.

3.3. Implementation of Pronoun Resolution

Pronoun resolution algorithms start from implementing the conditions in (5). Among the four conditions for each combinatorial rule, *crossover* and *slash-percolation* are common to all the three rules. Therefore the algorithms for these two conditions are considered first. (13) is for the former, and (14) is for the latter. Here, **avop** refers to an Attribute-Value Ordered Pair <PHON, CAT, AGR, LPS, SLASH>. (15) is the algorithm for non-coreference in (5ai). These three are just computational implementations of Chierchia's constraints. Because the algorithm for LPS-percolation is the same as that of slash-percolation except that LPS substitutes for SLASH, this paper will not describe algorithm for LPS-percolation specifically.

(13) Algorithm for Handling a Crossover

function boolean handle_crossover(**avop** s_1, **avop** s_2)
var
    set A, B, C, D;
begin
    A = SLASH(s_2);[9]
    B = LPS(s_1);[10]
    C = LPS(s_2);
    D = $A \cap (B \cup C)$;
    If D is $\emptyset$ **then return true else return false**;
end;

(14) Algorithms for SLASH-percolation

function set slash_percolation(**avop** s_1, **avop** s_2)
var
    set A, B, C;
begin
    A = SLASH(s_1);
    B = SLASH(s_2);
    C = $A \cup B$;
    return C;
end;

(15) Algorithms for Non-coreference

function boolean non_coreference(**avop** s_1, **avop** s_2)
var
    set A, B, C;
begin
    A = LPS(s_1);
    B = LPS(s_2);
    C = $A \cap B$;
    If C is $\emptyset$ **then return true else return false**;
end;

---

[9]SLASH(s) is the function that returns SLASH value of s.

[10]LPS(s) is the function that returns LPS value of s.

(16) is an algorithm for checking agreement in *number* and *gender* between a pronoun and its antecedent. Here three flags f_1, f_2, and f_3 are used. f_1 is for *number*, f_2 is for *gender*, and f_3 for the combination of f_1 and f_2. This is an implementation of (3a).

(16) Algorithm for Checking Agreement

```
function boolean check_agreement(agr s1, agr s2)[11]
var
    boolean f1, f2, f3;
begin
    f1 = false;
    f2 = false;
    if NMBR(s1)[12] ≈[13] NMBR(s2) then f1 = true;
    if GNDR(s1)[14] ≈ GNDR(s2) then f2 = true;
    f3 = f1 and f2;
    return(f3);
end;
```

Now, it is time to implement two combinatorial rules (5a) and (5b). (17) is for (5a) and (18) is for (5b). Note that all the conditions in (5a) and (5b) are included in body of the algorithms.

---

[11] **agr** *s* is the agreement features of *s*. The three matrixes in (4) are the examples.

[12] NMBR(*s*) is the function that returns *number* value of AGR value in *s*.

[13] ≈ means *compatible with*, which is similar to that of HPSG.

[14] GNDR(*s*) is the function that returns *gender* value of AGR in *s*.

(17) Algorithm for TV + NP ⇒ IV

```
function avop TV_NP(avop s1, avop s2)
var
    boolean f1, f2;
    avop s3;
begin
    PHON(s3) = PHON(s1) + PHON(s2);
    CAT(s3) = IV;
    AGR(s3) = AGR(s1);
    f1 = non_coreference(s1, s2);
    if f1 is true then
        begin
            SLASH(s3) = slash_percolation(s1, s2);
            LPS(s3) = lps_percolation(s1, s2);
        end;
    else
        return error;
    f2 = handle_crossover(s2, s3);
    if f2 is true then
        return s3;
    else
        return error;
end;
```

(18) Algorithm for $S/NP_n + NP_n \Rightarrow S$

```
function avop S/NP_NP(avop s1, avop s2)
var
    boolean f;
    string A, B;
    avop s3, s4;
    set S;
begin
    if INDEX(s1)[15] ≠ INDEX(s2) then
        return error;
    else
        begin
            A, B = separate_category(s1);
            PHON(s3) = PHON(s1) + PHON(s2);
            CAT(s3) = A;
            s4 = AGR(B);
            if check_agreement(s2, s4) then AGR(s3) = AGR(s2);
            LPS(s3) = Ø;
            SLASH(s3) = slash_percolation(s1, s2);
            S = LPS(s1) ∪ LPS(s2)
            if TYPE(s1)=refl then resolve_reflexive(s1,S);
            if TYPE(s1)=pron then resolve_pronominal(s1,S);
            f = handle_crossover(s2, s3);
            if f is true then
                return s3;
            else
                return error;
        end;
end;
```

Note that both reflexives and pronominals are resolved when S/NP_i and NP_i are combined. It is different from the analyses in (6) and (7). In Chierchia's analysis, after $(S/NP_2)/NP$ and NP_3 are combined, reflexive rule in (5d) is applied, which removes the reflexive index 3 from LPS and changes the translation $love'(x_3)$ into $\lambda x_3[love'(x_3)(x_3)]$. In its actual implementation, we

[15]INDEX(s) is the function that returns INDEX value of AGR in s.

will process these steps in one scoop, and handle it in the rule for $S/NP_n + NP_n \Rightarrow S$.

We will finish implementing resolution algorithms for English pronouns by providing concrete mechanisms for handling reflexives and pronominals. (19) is the algorithm for reflexives. Tough the algorithm itself is a little complicated, basic idea is simple. When we have a reflexive, we look into the NPs whose indexes are stored in LPS. For each NP in LPS, we check the agreement between the reflexive and the NP. If there is no conflict, we change the index of the reflexive into that of the appropriate NP. If not, we continue to search a next possible antecedent.

(19) Algorithm for Reflexive Resolution

```
function void resolve_reflexive(avop s1, set S)
var
    boolean f1, f2;
    int n;
    agr s2;
begin
    f1 = false;
    while all the indexes in S are not checked and f1 is false do
        begin
            n = the index;
            s2 = AGR(NPn);
            f2 = check_agreement(s1, s2);
            if f2 is true then
                begin
                    INDEX(s1) = n;
                    f1 = true;
                end;
        end;
    if f1 is true then
        begin
            S = S − INDEX(s1);
            print PHON(s1) + '=' + PHON(s2);
        end;
end;
```

(20) is the algorithm for pronominals. The resolution process is similar to that of reflexives. The difference is that the index of a pronominal must be different from that of an antecedent. Everything else is similar to the algorithm for reflexives.

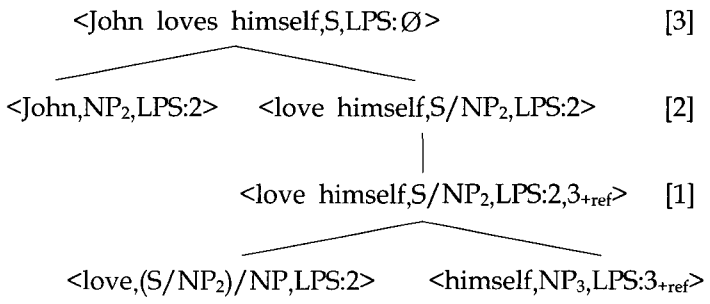
(20) Algorithm for Pronominal Resolution

```
function void resolve_pronominal(avop s1, set S)
var
    boolean f1, f2;
    int n;
    agr s2;
    avop s3;
begin
    f1 = false;
    while all the indexes in S are not checked and f1 is false do
        begin
            n = the index;
            if n ≠ INDEX(s1) then
                begin
                    s2 = AGR(NPn);
                    s3 = avop of s2;
                    f2 = check_agreement(s1, s2);
                    if f2 is true then f1 = true;
                end;
            else
                print PHON(s1) + '≠' + PHON(s2);
        end;
    if f1 is true then
        begin
            S = S − INDEX(s1);
            print PHON(s1) + '=' + PHON(s2);
        end;
end;
```

(21) and (22) are the sample analyses of reflexives and pronominals in this system respectively. In (21), [1] is the stage before the reflexive is resolved, [2] is that after reflexive resolution, i.e., after the output is generated, and [3] is that after

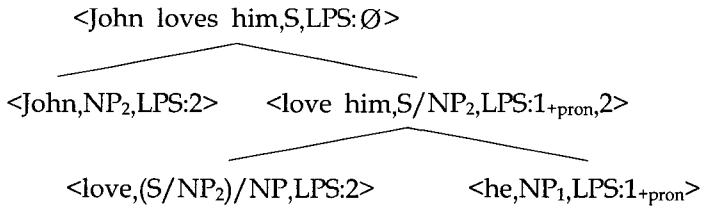
S/NP_2 and NP_2 are combined. Those two steps, from [1] to [2] and from [2] to [3], are handled within the same function in the algorithm of (19), though (21) illustrates them step by step.

(21) John loves himself.



Output : himself = John

(22) John loves him.



Output : him ≠ John

The differences between Chierchia's analyses in (6) and (7) and those in (21) and (22) are that (i) translation is not used in this CCG-like system, (ii) pronominals as well as reflexives have subscripted index type +pron, and (iii) for reflexives, two steps in Chierchias algorithm are processed at once in this system.

4. Advantages of Pronoun Resolution in Categorial Grammar

In this paper, we tried to develop pronoun resolution algorithms with a CCG-like system. We have roughly two advantages by using a CCG-like system.

The first one is that we don't need a parse tree to get pronoun readings. The systems that depend on structural configurations, such as Hong and Lee (1994), presuppose the existence of a parse tree. In this system, however, functional applications and possibly functional compositions take the roles of a parser, because all the operations are triggered with functor-argument relations. Therefore, we can get the same effect without a parser.

Second, we can speed up processing time. Because we can get the same effect of a parser without it, we dont need time for a parser, and it enables us to get fast processing time for pronoun resolutions.

5. Conclusion

We said that there are roughly two types of theories for pronoun resolution. One is to use tree structures, but it needs a parse tree, and requires much processing time. The other depends on functor-argument relations. This method doesn't require a parse tree, and we can achieve fast processing time.

This paper bases on Chierchia's (1988) Binding Theory in Categorial Grammar, and tries to implement his algorithms by a CCG-like system. Through the implementation, we saw that Chierchia's ideas are effectively implemented by the operations on attributes in categories.

In sum, this paper tries to give a faster and succinct computational algorithm for pronoun resolution in English,

within Categorial Grammar. I hope this study can give us a opportunity to understand pronoun system in English.

References

Ajdukiewicz, K. 1935. Die syntaktische konnexität. *Studia Philisophica* 1, 1-27.

Bach, E. and B. Partee. 1980. Anaphora and semantic structure. In K. Kreiman and A. Ojeda, eds., *Papers from the Parasession on Pronouns and Anaphora*. Chicago: CLS.

Chierchia, G. 1988. Aspects of a categorial theory of binding. In R. Oehrle et al. eds., *Categorial Grammars and Natural Language Structures*, 125-51. Dordrecht: R. Reidel.

Hong, S.-Sh. and Y.-H. Lee. 1994. On the implementation of reading pronominals using binding condition B. *Proceedings of Korean Cognitive Science*, 443-59.

Jurafsky, D. and J. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle, NJ: Prentice Hall.

Larson, R. 1988. On the double object construction. *Linguistic Inquiry* 19, 335-91.

Lee, Y.-H. in prep. Long-distance reflexives in Korean: a categorial grammar approach. Ms. University of Illinois, Urbana-Champaign.

Steedman, M. 1996. *Surface Structure and Interpretation*. Cambridge, Mass.: MIT Press.

Steedman, M. 2000. *The Syntactic Process*. Cambridge, Mass.: MIT Press.

Lee, Yong-hun
Dept. of Linguistics
4088 Foreign Language Building
707 S. Mathews Ave.
University of Illinois at Urbana-Champaign
Urbana, IL 61801
U.S.A.
Phone: +1-217-367-7261
E-mail: yleeuiuc@hanmail.net

접수일자: 2001. 8. 17.
게재결정: 2001. 11. 30.