「언어연구」, 제10집(1991), 193~225 *Linguistic Research, Vol.10*

A Gammar Model for Left-To-Right Parsing*

Gyonggu Shin, Kyung-An Song and Hwan-Mook Lee

1. Introduction

The grammar model we are proposing is a Linear Phrase Structure Grammar (hereafter LPSG), which can linearly generate and recognize sentences from left to right based on a set of phrase structure rules.[1] LPSG is an offspring of GKPS (Gazdar, Klein, Pullum and Sag 1985) in that it shares with them many properties including context-freeness, category concept as a feature bundle, monostratal construction. It is different from GKPS by utilizing no metarules, no trace symbol, and drastically different syntactic and semantic operations. It is also similar to the categorial grammar in that the syntactic rules are restricted to binary rules, and consequently binary composition of two categories. It differs from the categorial grammar by its use of ID rules, default categories, and its ability of linear parsing. **Default category assignment principle**, which is strictly ID-rule-dependent, will be extensively used to explain from simple sentence parsing to discontinuous dependencies. The major difference of LPSG from other grammar models is that it adopts the **strictly linear grammar** in which syntactic and semantic operations are homomorphic. The linearity of grammar is a consequence of extensive use of **default categories**, which is in turn made possible by constraining the grammar within the realm of **binary grammar**.

2. General Framework of LPSG

2.1 Mono-Stratal Syntax

For a grammar model to be a plausible candidate for a natural language, we

* The current idea is initially appeared on Shin (1987), modified in Myong, et al. (1988), and revised to be presented at the Kyung Hee International Conference on Linguistic Studies 1989.

1) It has nothing to do with the linear context free languages, in which the production rule are restricted to have no more than one non-terminal string to the right side of the rule as in A-〉 uBv or A -〉 u where both u and v represent terminal symbols.

2) For generation case is given right on the NP with no context in consideration. The speaker knows what case is necessary.

assume that it should explain some typical problems including cross-categorial generalization, cross-structural generalization, and unbounded dependencies, etc. In addition, it is expected to be efficient as a parser both in syntactic generation and semantic composition. This is why we adopt the monostratal syntax with no other structure than the surface. The surface structure is what the laymen speak, hear and think with. Even the trained linguists use the surface constructions when they are out on the street.

The transformational-generativism, while indulged in the complicated theorization, has been far away from ordinary people's intuition. However, the orientation set by Chomsky (1965) has contributed to the syntax by setting a correct direction: if a theory of grammar is explanatorily adequate, it should be psychologically realistic. The ruler of psychological reality finally has taken some of the syntacticians back to the starting point of mono-stratal perception of the language. Duo-stratal explanation of sentences forces a theory to be too complicated to be psychologically realistic: matching the surface structure with the deep structure is far more time-consuming than necessary (Forder 1983).

2.2 Extended Context-Free Grammar

In addition to the mono-stratal assumption, LPSG is considered to be context-free. The grammar is divided into four types based on the restriction levels: recursively enumerable grammar, context-sensitive (CS) grammar, context-free (CF) grammar, and regular grammar. It is a widely accepted assumption that human languages are at most in a recursively enumerable set. And we also assume that the grammar of human languages would not go below the power of the extremely restricted grammar of Type 3, regular grammar. Chomsky (1957: 19) proved the non-relevance of the Markov process to human language, a transition network equivalent to the regular grammar in its generative power. Our assumption on the scope of the natural language is that it would be located somewhere between Type O and Type 3 languages. And the continuous controversy is mainly on whether the CF grammar or the CS grammar would be more appropriate in describing the human languages.

It is commonly assumed that the CF phrase structure grammar (PSG) is considered too weak to describe and generate all the sentences of natural languages. Based on monodic category nodes, it loses cross-categorial generalizations as be-

tween verbs and adjectives even though both are considered as verbals. It does not capture cross-structural generalization on such constructions as exist between passives and actives, between normal and inverted sentences, between equi and raising constructions. It can not give an appropriate explanation on discontinuous dependencies. But both linguists and computer scientists have not stopped trying to find out ways to utilize the CF grammar because of its efficiency in parsing in comparison with the CS grammar, whose relative inefficiency in parsing has been a major reason of reluctance to accept it as a feasible grammar for human language (Back 1973: 195).

The expansion of the power of the CF PSRs was made possible by assuming the category symbol as a feature complex (GKPS 1985). And the expansion allows CF rules to represent relations between nodes. The expanded power of the CF PSRs is well represented in the **definite clause grammar**, which is similar to CF grammar. But the expressive power of its category symbols is expanded by making it have arguments equivalent to feature symbols. The revised grammar is implemented in PROLOG for natural language analysis (Pereira and Warren 1980).

The effect of revision is well-shown by the two different subject-verb agreement rules in (1a) and (1b), the former of which is traditionally considered to be treatable only by a CS rule (M. Gross 1972: 132f).

(1) a. Context-sensitive rule: VP → V[+Plu]/NP[+Plu]____

b. Context-free rules: S → NP[αPlu] VP[αPlu]

VP[αPlu] → V[αPlu] NP

The fact that the same category, say, a verb category functions in different ways as in (2) has been disappointing to those who want to render context-freeness to the natural language. Accordingly most of the PSRs of English are devised as CS rules as we see in (2). But we can contrive CF rules with subcategorized rule number as in (3) for each of the CS rules in (2). The CF rules in (3) are exactly equivalent in its generative power to those in (2) (Bach 1973; GKPS 1985: 35, 48; Church 1979: 92).

(2) V → V

V/__ NP

V/ __ NP NP
V/ __ NP to NP
...

(3) VP → V
VP → V[1] NP
VP → V[2] NP NP
VP → V[3] NP to NP
...

One of the main objectives of this paper is to limit the grammar generating English sentences within the realm of CF grammar with recognition process in mind. Note also that our rules will be different from the rules in (3). The examples in (3) only show a direction of the revision from the CS rules into CF ones.

2.3 Linear Grammar

We will go one step further from mono-stratal CF grammar to restrict our model within the scope of linear grammar in order to get to the psychological reality. This requirement derives from the consideration of the recognition process. Studies on the recognition process have already contributed to changing grammarians attitudes toward the PSRs, which is no longer considered as a generative process but as a node admissibility or well-formedness condition. We will take the rules to be linear-bound from left to right simply because we commonly do so while processing sentences. The linearity of the grammar gives rise to a further revision on the way of semantic translation: the translation is to follow the linear order of a sentence for the sake of homomorphism between syntax and semantics. And in contriving the way of logical translation, our prime concern is to be on the linear order of a sentence not on the sequence of composition from right to left in a traditional way.

The linear-ordered grammar from left to right is by no means newly-fashioned. It is all too common in the parsing strategies in the computer science. It is intuitively plausible as well. No speaker seems to generate and recognize sentences from right to left. Furthermore, no one seems to generate sentences in one way

(say, from left-to-right) and recognize them in the other way (say, from right-to-left). We will assume that a psychologically realistic grammar. should be in linear-ordered operation. A hearer often guesses what kind of constituent or word is coming next, while the other side is speaking. He frequently picks up a word or a phrase with a correct syntactic category, while the speaker is at a loss what word or phrase to select. This suggests to us that a human being is capable of parsing sentences in a linear order from left to right. And our basic assumption on the requirement of a grammar is linearity condition: any grammar of human language should explain the linear sequence of sentence generation and recognition.

2.4 ID Rules and LP Statements

We will decompose one PSR into two distinct parameters as in GPSG: an immediate dominance (ID) rule and a linear precedence (LP) statement. Rule (4) only states that VP is composed of V and NP in the given order. But with the LP statement absent, the expressiveness of the rule (5a) is expanded to the two PSRs in (5b).

(4) VP -⟩ V NP

(5) a. VP -⟩ V, NP
b. VP -⟩ NP V
VP -⟩ V NP

The simplifying effect is significant in the case of free word order languages such as Korean and Japanese. With the concept of normal PSR we need to have two different rules for transitive verbs, but these two rules in (6a) are reduced into one as in (6b) with LP statement factorized out as in (6c). The effect of the current revision will be easily noted with more nodes added before V, since the number of PSRs will be exponentially expanded in the case of (7).

(6) a. S -⟩ NP[NOM] NP[ACC] V
S -⟩ NP[ACC] NP[NOM] V
b. S -⟩ NP[ACC], NP[NOM], V

c. NP 〈 V

(7) S -〉 NP[NOM], NP[ACC], (PP), (ADVP), V

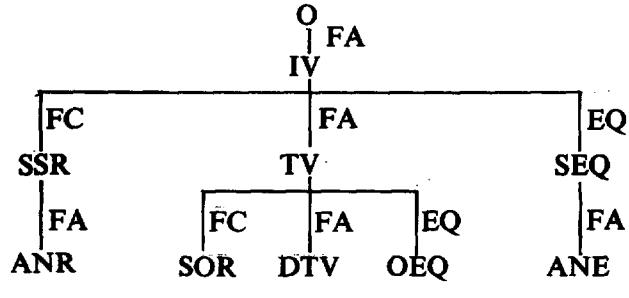
2.5 Category Levels

In order to make possible a linear-ordered operation, we adopt the category level which are similar to those of Pollard (1984:32), which again follows the theoretical construct of the categorial grammar. Consequently, X-bar will not be taken into our consideration. The phrasal bar level is not important in the binary combination of constituents.

Furthermore, we consider that X-bar syntax is not much of a generalization in the lexical verb categories. For example, kill and give, which have been considered to belong to the same bar-level, are different from each other in their way of combination. The former is combined with one objective NP(i.e. NP [ACC]) to be a VP, while the latter is combined with one NP (i.e. NP[DAT]) and again with another NP (this time NP[ACC]) to be a VP.

The different category levels decide its future career of composition. We will adopt the subcategorization schema of verbs given in (8). It shows how low-level subcategories of verbs are combined into a higher category level. FA represents a usual **functional application** between a functor and an argument, in which a functor is combined with an argument as in die '(John'). FC represnets a **functional composition** between one function and another to become a function as in (seem' (go')) (x). EQ indicates the case where an argument for a constituent IV is provided by the matrix subject or object NP. In (9) we will see a list of different subcategory types of lexical verbs. Note that when we use IV, it is a verb phrase which undergoes functional application to be a sentence.

(8)



(9)

Name	Description	Control type	Examples
O	no arguments	Null	S, NP, PP
IV	intransitive	⟨FA⟩	intr. verb, predicative
TV	transitive	⟨FA, FA⟩	TV, TV Phrase, preposition
SSR	subject-to subject raising	⟨FC, FA⟩	tend, certain, AUX
SEQ	subject equi	⟨EQ, FA⟩	try, eager (to help)
DTV	ditransitive	⟨FA, FA, FA⟩	give
SOR	subject to object raising	⟨FC, FA, FA⟩	believe (him to be honest)
OEQ	object equi	⟨EQ, FA, FA⟩	persuade (him to help)
ANR	anomalous raising	⟨FA, FC, FA⟩	seem (to be happy)
ANE	anomalous equi	⟨FA, EQ, FA⟩	promise (him to help)

NPs are also classified into different subcategorization as in the case of verbs. However, they differ from IVs. They have only three categorial levels: lexical category, semi-phrasal (NOM) category, and phrasal category.

2.6 Default Categories

GPSG utilizes syntactic features to such an extent that a categorial node is not monodic but a set of features. It does not only help capture the cross-categorial generalization, but its feature instantiation principles such as head feature convention (HFC) and foot feature principle (FFP) enrich the set of features inherited by the ID rules with functional informations, cases, tenses and even informations on displaced nodes and equi nodes. These features are classified into two: **HEAD** and **FOOT**. In this paper, we will exclude **slash** from FOOT, but expand the feature bundle with category-valued feature: **default category** as given in (10c). They are different from HEAD and FOOT features. They are operational features instantiated by default category assignment principle.

(10) a. HEAD = {N, V, PLURAL, PERSON, NOM, PFORM, AUX, ···, AGREEMENT, SUBCAT, LOCATIVE}
b. FOOT = {WH, REFLEXIVE/RECIPROCAL}
c. DEFAULT = {FILLED, LACKED}

The default category feature is an expansion of slash category, which is originally proposed in Bear (1982). The current default category is proposed to explain the linearity hypothesis of sentence analysis. The **filled category** is a category already processed syntactically. Since there is no gap in LPSG analysis of sentence structures, the filled category is not to be repeated in the sentence as in (11b). If it is repeated it is unacceptable.

(11) a. John, Mary loves.
b. *John, Mary loves John.

The **lacked category** is a category to be filled. For example, once we get a subject NP dominated by an S, an IV is obligatorily required to complete a sentence. And at the node of subject NP as in (12), we assume that an IV node is a lacked category. When we are on the node of be dominated by IV as in (13), ADJ node is required. We will use the set representation to show the lexical items which is included in a category as in both (12b) and (13b). A phrase or a sentence is legal as far as there is no lacked category.

(12) a. S → NP[NOM], IV
b. NP = {John, Mary, ···}
c. IV = {die, run, ···}

(13) a. IV → be, ADJ (be is an expression of IV[−AP] category)
b. be = {be, is, are, am, ···}
c. ADJ = {sick, happy, tired, ···}

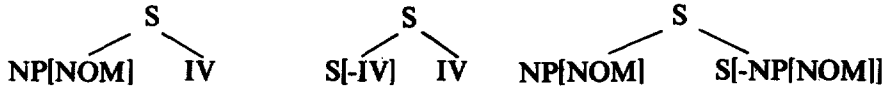
The default categories are easy to find, since they are provided by relevant ID rules. And it is not even necessary to specify which is filled or lacked since they are known to the language users. However, we will specify the lacked categories in this paper in order to show how lacked categories are expected and satisfied. That is why we call them simply **default categories**. For the language users, they are provided based on their intuition; in the professional viewpoint of grammarians, they are provided by ID rules. They are not a slash category in terms of the **filler-gap** relation; rather they are in the **gap-filler** relation. The slash category of GPSG is a lacked category in our LPSG. The former is filled by its preceding category while the latter is to be filled by its following category. While

the slash categories are commonly introduced by derived rules in GPSG, our filled/lacked categories are assumed to exist simply because of ID rules as the generalization in (14) on the default categories shows.

(14) Default categories

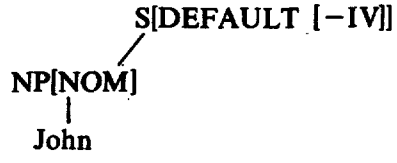
a. If there is a binary rule of the form X → Y, Z
then Y is equivalent to X[−Z], and
Z is equivalent to X[−Y].

b. Examples of default categories based on (12a):



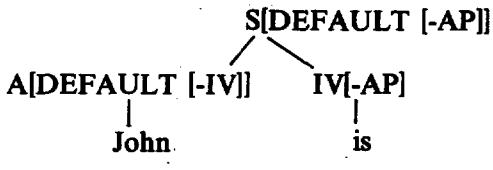
Based on the ID rules and lexical entries in (12) and (13), the process of generation and recognition begins as in (15). NP[NOM] is provided by the rule (12) and default category assignment. Unless a node is given, no information on a sentence is available, and no operation is initiated. After NP[NOM] is given, S node is assigned [DEFAULT [-IV]]. NP[NOM] is no longer expected to come since it is a filled category, and [IV] is expected to come since it is still lacked.

(15)



With a new lexical item is provided, the IV is partially completed but lacks AP, and we have the tree in (16). In the local level, the lacked category is provided by ID rule default based on ID rule (13). Leaving the local level, it is specified in the dominating IV node.

(16)



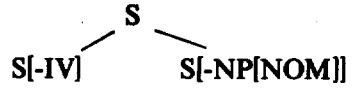
The left sister in a PSR can be considered as a category requiring its right sis-

ter. For example, be is considered as a category lacking PP[LOC], NP or Adj complement, and all the following three categories require be to precede them. An intransitive verb is a category which does not require any NP[ACC] but NP [NOM], while a transitive verb is a category requiring both NP[NOM] and NP [ACC], and so it goes on.

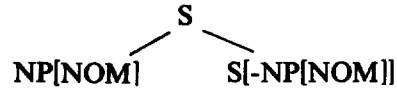
2.7 Default Category Assignment Principle

NP[NOM] is equivalent to S[-IV], and it needs to be combined with IV. But if we allow any category to be given default categories by ID rules without any constraint, the syntactic operation will become complicated and does not give much generalization. Syntactic composition of S[-IV] with S[-NP[NOM] in (17a) does not give any generalization. Composition of NP[NOM] with S[-NP[NOM]] is plausible, but it does not show linear operation: the operation is done backward.

(17) a.



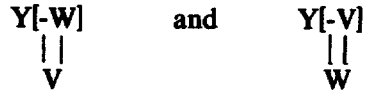
b.



Consequently, we need to establish a principle which governs the default category assignment as in (18). Double vertical bars between two nodes as in (18b) show that the two nodes are categorically identical and not in dominance relation.

(18) a. **Default Category Assignment Principle** (**DCAP** hereafter):
If there are binary rules of the form Y -> V, W
then V is equivalent to Y[-W], and
W is equivalent to Y[-V].

b. Tree representation of default category assignment:

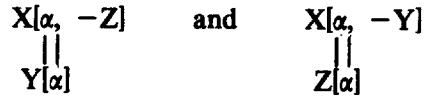


In order to show that the default categories can be percolated to the higher

node, we have another principle which governs the feature inheritance in a parse tree.

(19) a. **Default Category Inheritance Principle (DCIP hereafter):**
If there is a binary rules of the form X -› Y, Z
then Y[α] = X[α, −Z], and
Z[α] = X[α, −Y]
where α is a set of default categories.

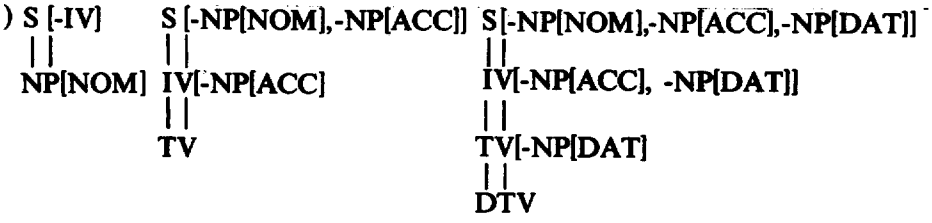
b. Tree representation of default category inheritance:



Note that DCAP is similar to the type-raising of the categorial grammar. In contrast to the type-raising, DCAP is strictly ID-rule-dependent. It is also simpler in operation and closer to what we think in syntactic operation in everyday use of language. Further examples of default category assignment are given in (20) and (21). The set of ID rules in (20) and DCAP with DCIP will give us parse trees in (21).

(20) S -› NP[NOM], IV
IV -› be, NP[ACC]
TV -› DTV, NP[DAT]

(21)



TV and IV[-NP[ACC]], in fact, are equivalent to S[-NP[ACC], -NP[NOM]]; and DTV to TV[-NP[DAT]] and IV[-NP[ACC], -NP[DAT]], and finally to S[-NP [ACC], -NP[DAT], -NP[NOM]]. Some of the readers might be worried about the infinite expansion of the number of default categories. But DCAP is ID rule-dependent, and thus the member of default categories will be properly constrained.

The constraint will be given further by composition principle in section 2.9.

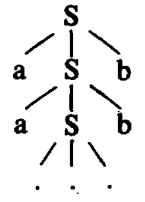
2.8. Binary Grammar

In order to utilize the default categories discussed in the previous section, we add a requirement for the linear-ordered grammar: the grammar consists of binary rules. In other words, the ID rules cannot bear more than two children; the third child is illegal. This binary property is also crucial in explaining the homomorphism between syntax and semantics, since the latter should be done through binary composition. First we will show that the binary grammar and the CF grammar can be equivalent in its expressive power.

There has been much effort to restrict the grammar in the realm of CF grammar, but no effort to confine it with the binary composition. Furthermore, incompatibility of syntactic and semantic operation is simply disregarded. It has been considered natural to carry out syntactic operation in one way and semantic operation in another. Even semantics-oriented GPSG constructs a semantic operation independent from syntactic operation. Following categorial semantics, we will consider the semantic operation should be binary. And then in order to get homomorphic operation between syntax and semantics, we should show that the power of the binary branching grammar can be equivalent to that of the normal CF grammar. Once we achieve this goal the grammar will be nearly as efficient as the regular grammar. One of the most common examples of CF grammar is the following one, which shows a string with a recursively infinite number of embedded strings of equal number of a's and b's: $a^n b^n$

(22) a. S -> aSb
S -> ab

b.

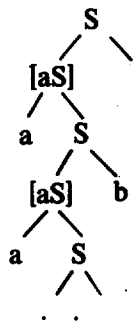


We can devise a binary branching grammar without reducing the power of a normal CF PSG with expanded function of nodes. The strings generated by the rules given above can also be generated by the following set of binary branching rules by introducing a new node of [aS] representing the sequence of a and S.

The only difference between the former and the latter grammars is that the latter can have one more string of a, which the former does not have. With another rule 'S → a' added to the binary grammar, the two become completely equivalent.

(23) a. S -› [aS] b
[aS] -› a S
S -› a

b.



The revised version generates a string of equal number of a's and b's. This is an important step toward having a syntactic operation with homomorphic semantic translation, for the semantic translation is carried out binarily.

2.9 Composition Principle

During the tree building operation we assume the composition principle of (25) is working. Filled categories are assumed to exist though it is not represented in the tree. The overt representation of the filled categories are indispensable for the linear sentence processing. They, however, will be represented in the semantic representation since required for semantic operation. In the following tree we will not give a special attention to the topicalized sequence of NP[ACC] followed by NP[NOM], since it is considered to be decided upon by the LP statements of (24). For the details of the LP statements refer to GKPS (1985: 248), though the LP statements should be further elaborated in terms of LPSG.

(24) LP statement:
a. [SUBCAT] 〈 [-SUBCAT]
b. [+N] 〈 PP 〈 VP
c. NP[ACC] 〈 S[-NP[ACC]]

S[-NP[ACC]] 〈 NP[ACC]

(25) Category Composition Principle (CCP hereafter):

a. **DCAP and DCIP applied to the first node**: A sentence initial node is assigned default categories up to the level of S.

b. **DCAP and DCIP applied to the non-first node**: When composition is not possible for a non-first node, it is assigned default categories to the level expected by the preceding node.

c. **Matching with an expected node without default categories**: When a lacked feature [-X] hits an X node with no dafult category, it is canceled.

d. **Matching with an expected node with default categories**:
When a lacked feature [-X] hits an X node with default categories, it is replaced by the default categories.

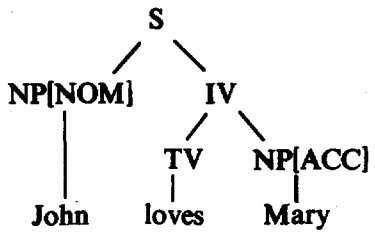
e. **Optional category**: When an optional category is expected, it may not be satisfied.

Strictly speaking the first two principles are further elaboration of DCAP and DCIP: they shows an environment where DCAP and DCIP apply. CCP in (25c) is similar to the functional application (FA) of categorial grammar: a functional category with a lacked argument combines with an argument category. CCP in (25d) is similar to the functional composition (FC) of categorial grammar: a functional category with a lacked argument combines with another functional category with a lacked argument.

CCP(a) states that, provided with a node at the beginning of a sentence, DCAP and DCIP is activated to introduce a default categories for the first node to be prepared to be combined with its following nodes somewhere in the sentence. CCP(b) says that DCAP and DCIP is activated to introduce a category which is able to be combined with its preceding as well as following categories.

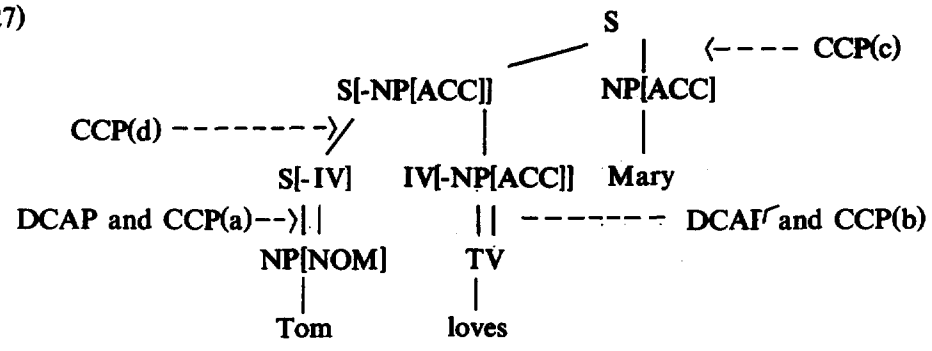
With the DCAP and CCP we can achieve syntactic linearity from left to right. Traditionally the generation is considered to be from right to left by top-down generation, while the semantic translation from right to left by bottom-up operation, which exactly corresponds to the traditional tree diagram in (26).

(26)



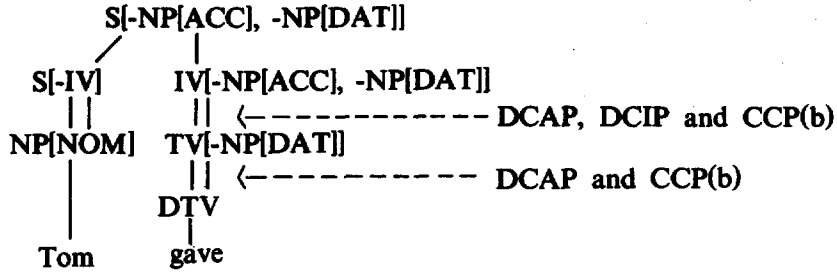
This tree does not explain left-to-right linear-ordered operation. So we revise the tree diagram into the one which will allow node combination from left to right, utilizing DCAP and CCP. The tree will be made through left branching operation instead of right branching for the sake of linearity. The tree in (26) will then be revised into the tree in (27). We will assume the tree not to be a final product, but a process from the left-most node to the right-most node.

(27)



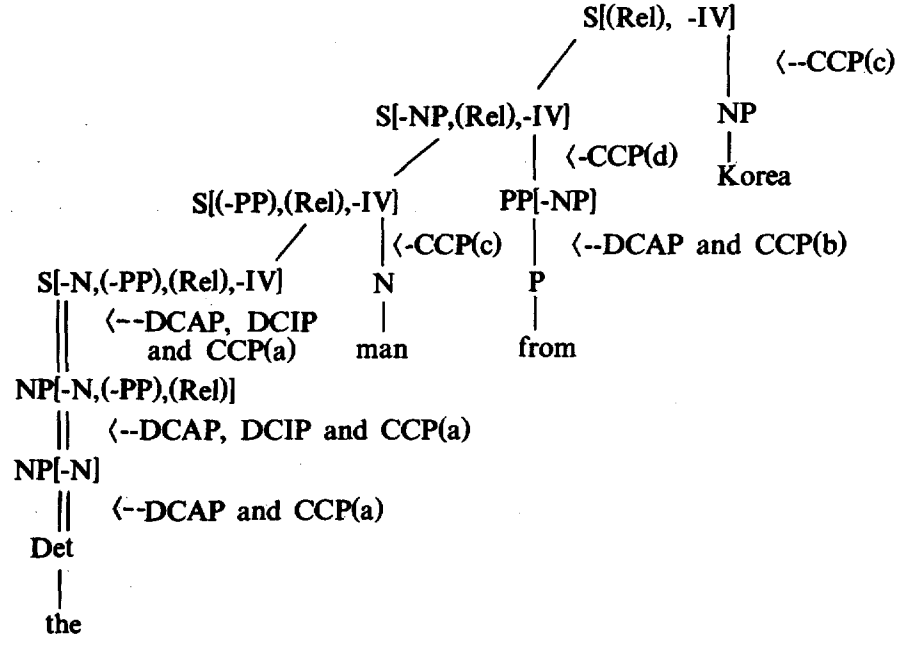
In some cases the DCAP can be applied multiply by CCP(b) until CCP is able to combine two nodes as in (28), where the DTV node is changed into TV and again IV to be combined with its preceding lacked category [-IV].

(28)



CCP(e) is necessary for the trees as in (29), where an optional category is possible for an NP node. Some phrases such as PP, relative clause can be post-positioned at the end of NP. We do not want to erase the unsatisfied optional phrases after NP, since they may not be satisfied at all or may be satisfied at the end of the sentence as in The death of a student who had been wanted by the police remains mystery.

(29)



As we see in the above operation there is no mechanism whatsoever for trace or slash categories to be introduced. Introducing an empty node is equivalent to saying that there is a rule for generating a null string. If we want to keep our grammar within the boundary of context-freeness, it is required that we avoid rules generating a null string. Note that there is no CF or CS rule which is capable of generating a null string. Only Type O grammar is able to have a rule generating a null string.

3. Applications on English Sentences

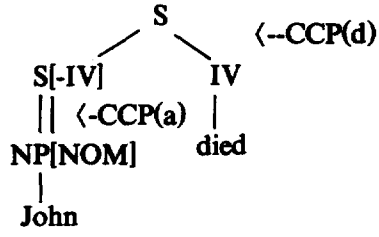
In this section we will see how LPSG works with some basic constructions of English: first IV, TV, and DTV, and then discontinuous dependencies. In the course of syntactic analysis, new category labels will be introduced, but they are not fixed ones yet.

3.1 Intransitive Verbs

IV construction may be considered to be generated by a unit production, in which there is only one daughter node. But we will accept only the rule in (30a). The tree diagram in (31) represents IV construction based on rule (30a).

(30) a. S -› NP, IV
b. IV = {follow, rise, die, run, ···}

(31)

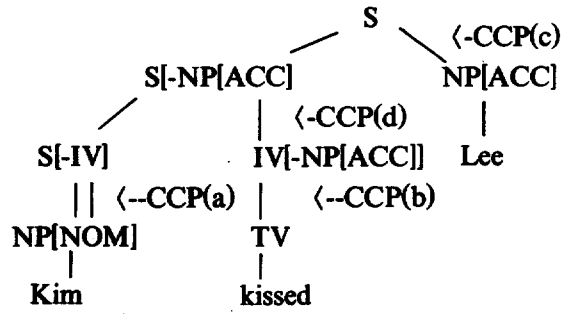


3.2 Transitive Verbs

Traditionally a transitive verb is combined with an accusative NP to be an IV as in (32c), which is in turn combined with a nominative NP to be a sentence. In LPSG, NP[NOM] is first combined with the transitive verb. The result of this linear-ordered operation, NP[NOM] followed by a transitive verb, is assigned a syntactic category of S[-NP[ACC]] by default category assignment principle as in (32d). S[-NP[ACC]] is then combined with NP[ACC] to be a sentence. The process of combination for a simple example is given in (32).

(32) a. IV → TV, NP[ACC]
b. TV = {kiss, love, close, ···}
c. __kissed Lee S[-NP[NOM]] or IV
d. Kim kissed__ S[-NP[ACC]]

(33)



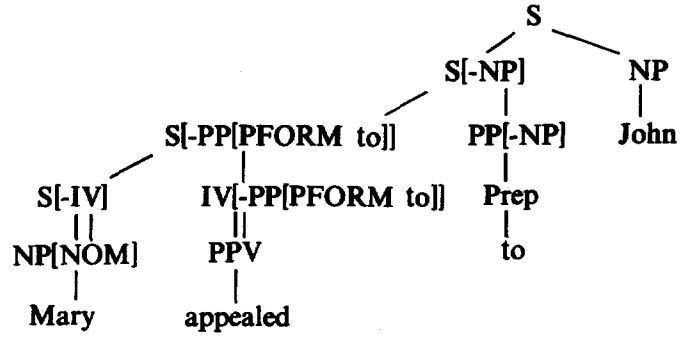
3.3 PPV: Verb Requiring PP

There is a category of lexical PPV which requires PP to become a IV. For example, apply in (34) does not become an intransitive verb by itself without PP [to] following. In line with this, we construct ID rules by separating intransitive verbs which require PPs from normal ones. It seems to be a misnomer to call the former either an intransitive or a transitive verb, since it differs from both in its way of composition: it needs a PP to become IV and then an NP[NOM] to become a complete S, while normal intransitive verbs need only one NP, and TV needs NP[NOM] and NP[ACC] to become a complete S. The feature of PFORM will be useful in (34) to distinguish what kind of PP they need to be followed by. There are five or more kinds of PFORMs in English. But we are going to deal with two of them here.

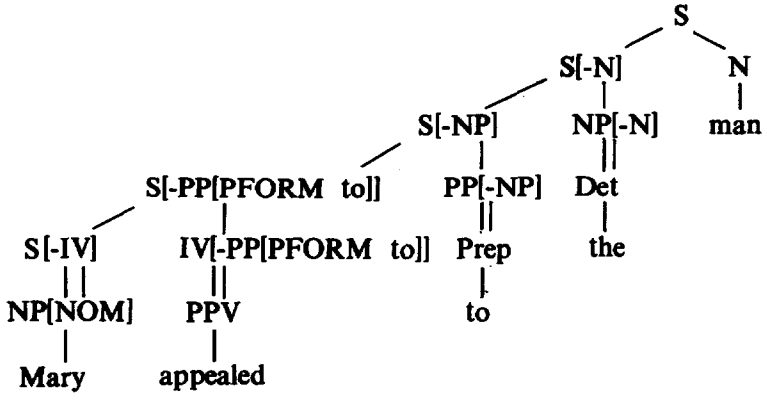
(34) a. IV → PPV[PFORM α], PP[PFORM α]
b. PPV[PFORM to] = {appeal, apply, contribute, ···}
PPV[PFORM for] = {apply, sing, ···}
c. __appealed to John S[-NP[NOM]]
__sang for John S[-NP[NOM]]
d. Mary appealed__ S[-PP[PFORM to]]
Mary sang__ S[-PP[PFORM for]]

If we strictly follow a strict linearity we have to have a tree diagram as in (35). When a sentence is relatively short as in (35a), the strict linearity seems to be quite efficient. But as a phrase such as PP phrase becomes longer, a strict linearity as in (35b) might become inefficient for parsing. And we may well qualify the requirement of the linearity of a grammar to the relative linearity in terms of the maximal category, which idea was diagrammed in (36). The superiority of efficiency between strict linearity and modified one is a matter of controversy which is not easy to be decided upon. And we want that question to stay undecided.

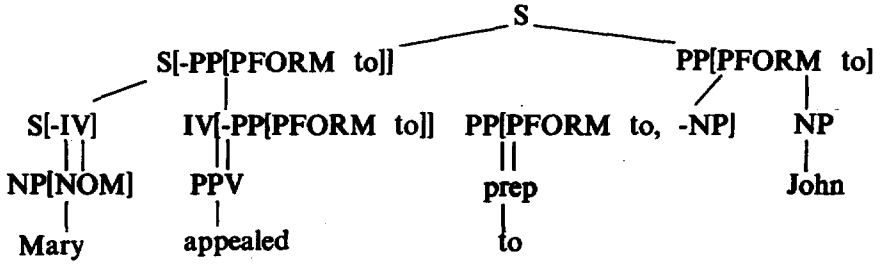
(35) a.



b.



(36)



3.4 Ditransitive Verbs

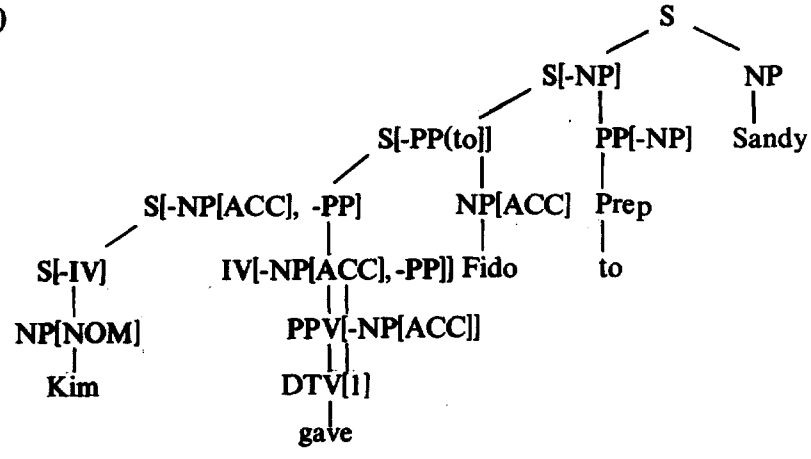
Ditransitive verbs or DTV is different from other transitive verbs by the fact that it combined with NP[ACC] to be a PPV and with NP[DAT] to be a TV. The syntactic difference between the two kinds of DTV can be represented with subcat number as GKPS(1985) did.

(37) a. PPV[1] → DTV[1], NP[ACC]
b. DTV[1] = {give, introduce, ···}
c. __gave Fido__ S[-NP[NOM], -PP[to]]
d. __gave Fido to Sandy S[-NP[NOM]]
e. John gave__ __ S[-NP[ACC], -PP[to]]
f. John gave Fido__ S[-PP[to]]

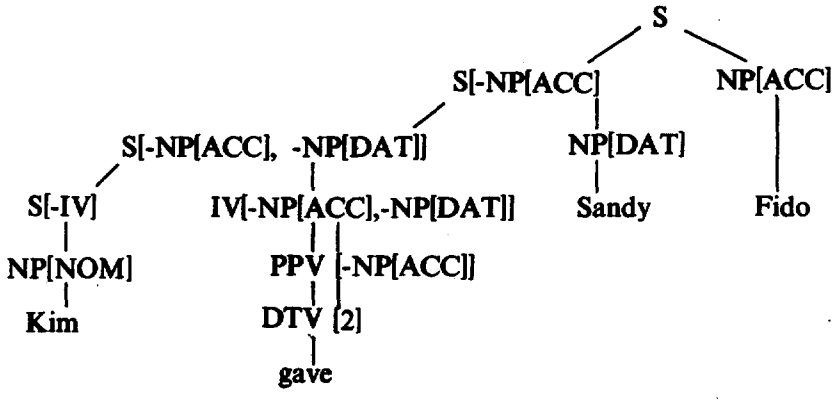
(38) a. TV → DTV[2], NP[DAT]
b. DTV[2] = {give, send, ···}
c. __gave Sandy__ S[-NP[NOM], -NP[ACC]]
d. __gave Sandy Fido S[-NP[NOM]]

The order of the two objects may be explained by LP statements of either NP ⟨ PP in the case of DTV[1] or NP[DAT] ⟨ NP[ACC] in the case of DTV[2]. The example tree in (39) is formed from the ID rules in (37), and tree in (40) from the ID rules in (38).

(39)



(40)



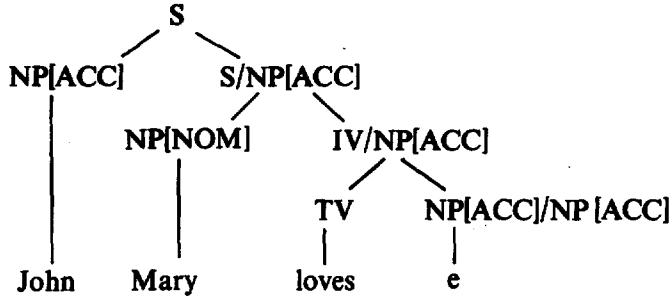
We assume that the category ambiguity of gave is not solved until it hits Sandy and Fido. The two possibilities are either retained until the parallel parser hits Sandy and Fido, or the parser back-tracks to find the correct recognition. The matter of superiority between parallel parsing and backtracking will be left open-ended again for the later discussion.

3.5 Topicalization

One of the important by-products of the current use of DCAP is that we do not need the slash termination metarule (STM) as used in GKPS. Since our grammar does not generate a trace in the tree, there is no need of slash termination metarule to get rid of it. A topicalized sentence needs to have a empty node in GKPS(1985) as in (42), since it is generated from the rules in (41).

(41) a. S -› NP[NOM], IV
b. IV -› TV, NP[ACC]

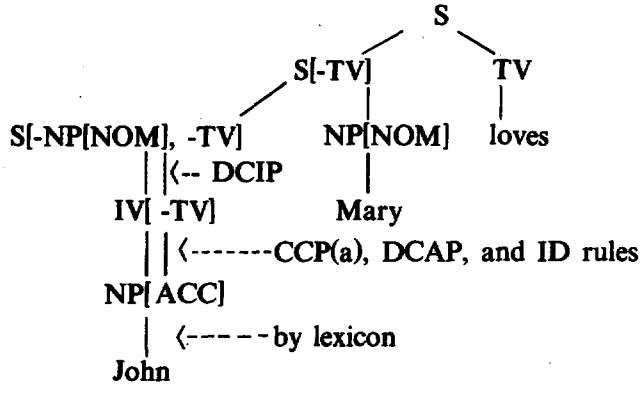
(42) a.



LPSG with default categories does not bear a child with empty node. A filled category simply is not expected to come. If any unexpected node appears without being able to be combined with any preceding lacked category, the sentence overflows with the unnecessary node and is judged to be unacceptable. On the other hand, if an expected node does not appear, the sentence is incomplete with the expected node unsatisfied, and it is also judged to be unacceptable.

Example tree in (43) explains how LPSG operates on the topicalized construction. Additionally, LP statement will deal with the linear order of the topicalized object NP, which is similar to that of GKPS.

(43)



3.5 Crossing

We already dealt with the discontinuous dependency construction with topicalization.. In this section we will see how the crossing construction is treated with in LPSG. Discontinuous dependency of the prepositional particle and the main verb which are separated by an accusative NP as in (44) is one of the most difficult problems to be solved with the CF rules. Here again our concern will be confined in syntax. Traditionally the two-word verbs are considered to be best explained with transformation, which asserts that a transformational rule moves the verb particle such as up, on over the following NP. But we will assume that there is no movement rule for the particles, and the verb particle (Part in this paper) is considered as a lacked default category to be satisfied somewhere later as in the rules (45). Part is dominated by TV in (45b), and preceded by NP [ACC] as in (46).

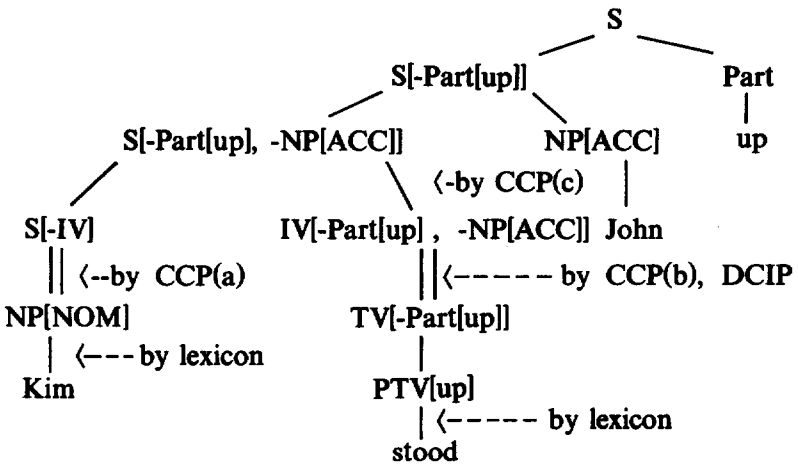
The tree diagram of (47) is based on the rules in (45) and (46).

(44) Kim stood John up.

(45) a. IV -› TV, NP
b. TV -› PTV, Part
c. PTV -› {stood[PForm up], call[PForm up],···}
d. Part -› {up, on,···}

(46) LP Statement: NP 〈 Part

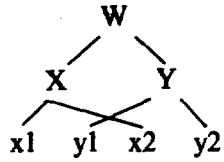
(47)



The basis of the current treatment is that the verb stand is not a usual kind of transitive verb, but it is a different class of transitive verb which requires a prepositional particle after it. In other words it is a verb which is to be followe by a prepositional particle and an accusative NP.

Following the same way of theorization as in the two-word verbs, the double crossing constructions are dealt with in LPSG. The double-crossing construction of (48) will be revised into ID rules and LP statement in (49) and (50), respectively. Based on the ID rules in (49) and LP statements in (50), we can construct a tree as in (50) with the help of default category assignment principle.

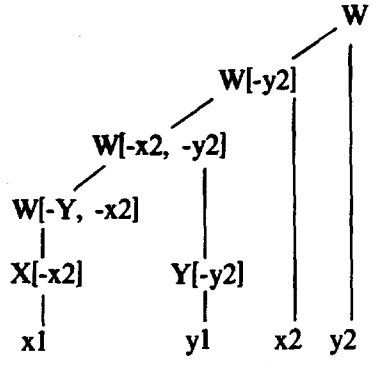
(48)



(49) a. W -› X, Y
b. X -› x1, x2
c. Y -› y1, y2

(50) LP statements:
a. X ⟨ Y
b. x1 ⟨ y1 ⟨ x2 ⟨ y2

(51)



4. Semantic Analysis of LPSG

The semantic analysis of LPSG will follow the basic ideas of Montague Grammar (MG hereafter), and it will be implemented by Prolog. In the computer implementation we follow the idea of Pereira & Shieber, (1987), who use the symbol "^" a built-in predicate of Prolog, as lambda-operator (cf. Monague 1970, 1973). The lambda calculation of Prolog as shown in Pereira & Shieber is different from and more powerful than that of the classical MG in several points. First, MG, which is based on the higher order logic, can have variables of different types, while the Prolog, which is based on the first order logic, can have variables of a single type. For example, MG can have three different types of variable for semantic analysis of (52a), and be in (52a) is translated as in (52b) (Dowty 1979:

364).

(52) a. John is old.
b. λP λx P[x]

The same representation can be utilized in Prolog but in a quite different sense: the variable x in (52b) would be considered as a constant because it begins with a lower-case letter. If it is written in capital letter, it will be considered as a variable of the same type as P. It is certain that this limitation within one type of variable is a severe restriction to the semantic analysis of natural language. But it is overcome in Prolog by the use of lambda operator. The verb be si translated in this framework as in (53), and proper nouns and determiners, which contain second order variables in Montague semantics, are translated as in (54).

(53) be =⇒ copula((X^P)^X^P)

(54) a. John ==⟩ np((john^S)^S)
b. every ==⟩ det((X^P)^(X^Q)^all(X, P =⟩ Q))
c. a ==⟩ det((X^P)^(X^Q)^exist(X, P & Q))

A further difference of the Prolog lambda calculator from MG is that the lambda operator can be combined with any category variable. The semantic translation of the sentence in (52a) is made possible by the phrase structure rules given in (55). In (55), the lambda operator gives rise to VP with the argument of ADJ, which kind of operation is impossible in MG. In fact "NP^S" and "ADJ^S" are variable combinations of the same type in Prolog.

(55) a. s(S) → np(NP), vp(NP^S).
b. vp(VP) → copula(ADJ^VP), adj(ADJ).
c. np(john) → [john].
d. adj(X^old(X)) → [old].

The third characteristics of Prolog lambda calculation is lambda fronting: a variable abstracted by the lambda operator can be abstracted again, which is represented in the formula of MG in (56).

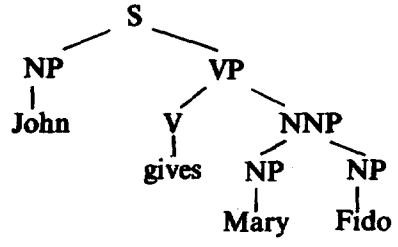
(56) a. John loves Mary.
b. John → λPP(john)
c. Mary → λQQ(mary)
d. love → λxλy love(x, y)
e. loves Mary →
λQQ(mary)(λxλy love(x, y)) → functional rule
λx[λQQ(mary)(λy love(x, y))] → lambda fronting
λx[λy love(x, y)(mary)] → lambda conversion
λx[love(x, mary)]
f. John loves Mary →
λPP(john)(λ x [love(x, mary)]) → functional rule
λx[love(x, mary)](john) → lambda conversion
love(john, mary)

The basic expressions in (56b) through (56d) are not calculable in MG because of the different types of the variables. But the lambda fronting solves the type problem of MG as we see in Prolog program (57).

(57) a. s(S) → np(VP^S), vp(VP).
b. vp(X^S) → tv(X^VP), np(VP^S).
c. np(john^S)^S) → [john].
d. np((mary^S)^S) → [mary].
e. tv(X^Y^love(X, Y)) → [love].

The fourth characteristics of Prolog lambda calculation is lambda postponement: the arguments can be stored to be calculated later once for all as we see in (58b).

(58) a. John gives Mary Fido.
b.



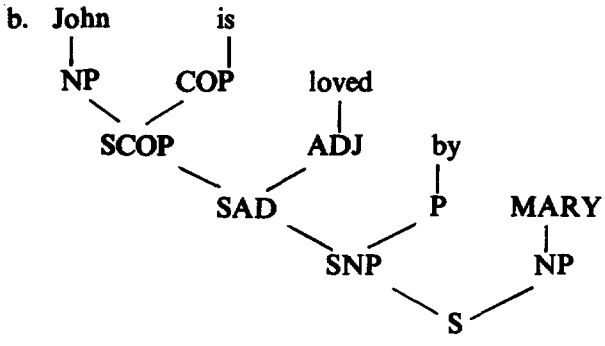
The analysis in (58b) is implemented into Prolog program as in (59). The process of semantic composition by Prolog is given in (60). The string of Mary Fido is given a category name nnp, temporarily.

(59) s(S) → np(NP), vp(NP^S).
vp(X^VP) → dtv(X^DTV), nnp(DTV^VP).
nnp((NP1^NP2^NNP)^ NNP) → np(NP1), np(NP2).
np(john) → [john].
np(mary) → [mary].
np(fido) → [fido].
dtv(X^Y^Z^ give(X,Y,Z)) → [give].

(60) ?− np(LF, [mary] []).
LF = mary →
yes
?− nnp(LF, [mary, fido], []).
LF = (mary ^ fido ^ _R0115) ^ _RO115 →
yes
?− vp(LF, [give, mary, fido], []).
LF =_R0105 ^ give(_RO105, mary, fido) →
yes
?− s(LF, [john, give, mary, fido], []).
LF = give(john, mary, fido) →
yes
?−

In LPSG the passive sentence (61a) is analyzed as in (61b). The syntactic categories are assumed to be calculated according to the principles in previous sections. For convenience' sake we will use the mnemonic category names in this section.

(61) a. John is loved by Mary.



We analyze the passive participle as adjective and the preposition "by" as a normal preposition (Song 1985 and 1986). The preposition "by" is translated as in (62). And the Prolog program for semantic analysis of (61a) is given in (63). Proper nouns are analyzed as e-type expression here. Some more example programs for the linear semantic analysis are given in the appendix.

(62) by ==⟩ P ^ X ^ (P & AGENT(X))

(63)
```
scop(A,B,C):-np(D,B,E), cop(D ^ A,E,C).
sad(A,B,C):-scop(D,B,E), adj(D ^ A,E,C).
snp(A,B,C):-sad(D,B,E), p(D ^ A,E,C).
s(A,B,C) :-snp(D ^ A,B,E), np(D,E,C).
np(john, [john:A],A).
np(mary, [mary:A],A).
cop(A ^ A,[is:B],B).
adj(A ^ exist(B,love(B,A)), [loved:C],C).
p(A ^ B ^ (A & ag(B)),[by:C],C).
```

5. Conclusion

The current work began with the belief that sentences should be processed in the linear order from left to right, with homomorphic operation between syntax and semantics in order for a grammatical theory to be psychologically realistic. We have constructed such a grammar by binary rules with extensive use of default category assignment principles and composition principle. At the bottom of

this whole idea lies the expansion of the expressive power of category by syntactic features especially by default category feature. The formalization is complicated, but it is still a CF grammar. And furthermore it does not imply any complication of the process of both generation and recognition, for it is possible to devise a CF grammar which is nearly as simple as a regular grammar and which is operable in left-to-right linear operation as well as homomorphic between syntax and semantics.

The current binary linear-ordered grammar is far from being complete as is presented in this paper. It does not say much about some typical issues such as the complex NP constraint, the left branching constraint, parasitic gaps, and the double hole constraint. We merely dealt with a few examples to show its expressive power.

References

Aho, A.V. and J.D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling.* Prentice-Hall, Englewood Cliffs, NJ.

Bach, E. 1973. *Syntactic Theory.* Holt Rinehart and Winston, NY.

Bear, J. 1982. *Gaps as Syntactic Features.* IULC.

Chomsky, N. 1957. *Syntactic Structures.* Mouton, The Hague.

——. 1965. *Aspects of the Theory of Syntax.* The MIT Press.

——. 1975. *Logical Structure of Linguistic Theory.* Plenum, NY.

Church, K.W. 1979. *Phrase Structure Parsing.* IULC.

Dowty, D. 1979. *Word Meaning and Montague Grammar.* Reidel, Dordrecht.

——. 1988. Type Raising, Functional Composition, and Non-Constituent Conjunction. in Oehrle et. al. 1988. eds. *Categorial Grammars and Natural Language Structures.* Reidel, Dordrecht.

Gazdar, G. 1982b. Phrase Structure Grammar. P. Jacobson and G. K. Pullum, eds. *The Nature of Syntactic Representation.* Reidel, Dordrecht, 131−186.

Gazdar, G., E. Klein, G. Pullum and I. Sag 1985. *Generalized Phrase Structure Grammar.* Harvard Univ. Press.

Gross, M. 1972. *Mathematical Models in Linguistics.* Prentice-Hall, Englewood Cliffs, NJ.

Hopcroft, J.E., and J. D. Ullman 1969. *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, MA.

——. 1979. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley,

Reading, MA.

Montague, R. 1970. Universal grammar, in Thomason. R. 1974. ed. *Formal Philosophy,* The Yale University Press, London.

Montague, R. 1973. The proper treatment of quentification in ordinary English. in Thomason, R. 1974. ed. *Formal Philosophy.* London:The Yale University Press, London.

Myong Ro-Keun, Park Hyo-Myong, Shin Gyonggu. 1988. Linear Phrase Structure Grammar. *Language Research* 24:2. Seoul National University Press.

Oehrle, R., E. Bach, D. Wheeler. 1988. eds. *Categorial Grammars and Natural Language Structures.* Reidel, Dordrecht.

Pereira, F.C.N. and D.H.D. Warren. 1980. Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13.

Pereira, F. & S.Shieber. 1987. *Prolog and Natural-Language Analysis. CSLI,* Stanford University.

Peters, S. and R. Ritchie. 1973. On the Generative Power of Transformational Grammar. *Information Sciences* 6, 49-84.

Pollard, C.J. 1984. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language.* Ph. D. dissertation, Stanford University.

Pollard, C.J. and I. Sag 1987. *Information-based Syntax and Semantics I.* CSLI, Stanford University.

Pullum, G.K., and G. Gazdar 1982. Natural Languages and Context-free Languages. *Linguistics and Philosophy* 4, 471-5—4.

Shieber, S. M. 1986. An Introduction to Unification-based Approach to Grammar. CSLI, Stanford University.

Shin, G. 1987. *A Phrase Structure Approach to English Syntasx.* Ph. D. dissertation, Chonbuk National University.

Song, K. 1985. A lexical approach to English passive. in Lee, I. ed. *Seoul Papers in Formal Grammar Theory.* Hanshin Publishing Co., Seoul.

Song, K. 1986. *Passiv: Seine Form und Funktion.* Doctoral dissertation of Ruhr-University Bochum, W/Germany.

Wall, R. 1972. *Introduction to Mathematical Linguistics.* Prentice-Hall, Englewood Cliffs, NJ.

Winograd, T. 1984. *Languages as a Cognitive Process.* Addison-Wesley, MA.

Appendix: Prolog programs for the linear semantic analysis of some example sentences in English.

Example 1: A man walks.

```
s(A,B,C): -np(D^A,B,E), v(D,E,C).
np(A,B,C): -det(D^A,B,E), n(D,E,C).
det((A^B)^(A^C)^exist(A.B & C),[a:D],D).
n(A^man(A),[man:B],B).
v(A^walk(A),[walks:B],B).
```

Example 2: John is old.

```
s(S) → scop(SCOP), adj(SCOP^S).
scop(SCOP) → np(NP), cop(NP^SCOP).
np(john) → [john ].
cop(NP^NP) → [is ].
adj(X^old(X)) → [old ].
```

Example 3: John is a boy.

```
scop(A,B,C):-pn(D,B,E), cop(D^A,E,C).
snp(A,B,C):-scop(D,B,E), det(D^A,E,C).
s(A,B,C):-snp(D^A,B,E), n(D,E,C).
pn(john,[john:A ],A).
cop(A^B^(B=A), [is:C], C).
det((A^B)^(A^D)^exist(A, B&D), [a:E], E).
n(A^boy(A),[boy:B ],B).
```

Example 4: Every man is a woman.

```
np(A,B,C):-det(D^A,B,E), n(D,E,C).
sv(A^B,C,D):-np(E^B,C,F), v(A^E,F,D).
sva(A,B,C):-sv(D,B,E), det(D^A,E,C).
s(A,B,C):-sva(D^A,B,E), n(D,E,C).
det((A^B) (A^C)^all(A,B => C),[every:D ],D).
det((A^B) (A^C)^exist(A,B & C),[a:D ],D).
n(A^man(A),[man:B ],B).
n(A^woman(A),[woman:B ],B).
v(A^B^(B=A),[is:C ],C).
```

Example 5: John loves Mary.

```
stv(A,B,C):-np(D,B,E), tv(D^A,E,C).
s(A,B,C):-stv(D^A,B,E), np(D,E,C).
tv(A^B^love(A,B),[loves:C ],C).
np(john,[john:A ],A).
np(mary,[mary:A ],A).
```

Example 6: Every man loves a woman

```
np(A,B,C):-det(D^A,B,E), n(D,E,C).
sv(A^B,C,D):-np(E^B,C,F), v(A^E,F,D).
sva(A,B,C):-sv(D,B,E), det(D^A,E,C).
s(A,B,C):-sva(D^A,B,E), n(D,E,C).
det(A^B)^(A^C)^all(A,B ⇒ C),[every:D],D).
det((A^B)^(A^C)^exist(A,B & C),[a:D],D).
n(A^man(A),[man:B ],B).
n(A^woman(A),[woman:B ],B).
v(A^B^love(B,A),[loves:C ],C).
```

Example 7: John gives Mary Fido.

```
s(S)→ sdio(NP^S),. np(NP).
sdio(SDIO) → sdi(NP^SDIO), np(NP).
sdi(SDI) → np(NP), dtv(NP^SDI).
np(fido) → [fido ].
np(mary) → [mary ].
np(john) → [john ].
dtv(X^Y^Z^give(X,Y,Z)) → [gives].
```

Example 8: John, Mary loves.

```
nnp((A^ B^C)^ C,D,E):-np(A,D,F), np(B,F,E).
s(A,B,C):-nnp(D^A,B,E), tv(D,E,C).
np(john,[john:A ],A).
np(mary,[mary:A ],A).
tv(A^B^ love(B,A),[loves:C ],C).
```

Example 9: Mary, John gives Fido.

```
nnp((A^B^C)^C,D,E):-np(B,D,F), np(A,F,E).
```

```
nnptv(X^A,B,C):-nnp(D^A,B,E), dtv(X^D,E,C).
s(A,B,C):-nnptv(D^A,B,E), np(D,E,C).
np(john,[john:A ],A).
np(fido,[fido:A ],A).
np(mary,[mary:A ],A).
dtv(A^B^C^give(B,A,C),[give:D ],D).
```

Example 10: John is loved.

```
scop(SCOP) → np(NP), cop(NP^SCOP).
s(S) → scop(SCOP), adj(SCOP^S).
np(john) → [john ].
cop(NP^NP) → [is ].
adj(X^exist(Y, love(Y, X))) → [loved ].
```

Chonnam National University