KORPAR: A RULE-BASED DEPENDENCY PARSER FOR KOREAN IMPLEMENTED IN PROLOG

by

SOYOUNG KWON

(Under the Direction of Michael A. Covington)

ABSTRACT

Natural language parsing is the process of analyzing an input sentence by determining its syntactic structure and representing that structure according to a given formal grammar. However, it is often difficult to parse sentences correctly since the nature of language is ambiguous and has many irregularities. If the word order is totally or partially free, the task of parsing becomes more challenging. The process of parsing can be based on hand-coded heuristic rules, probability, or a hybrid of both.

KorPar, described in this dissertation, is a parser for Korean based on hand-coded heuristic rules, represented in unification-based dependency grammar and implemented in Prolog. The dependency grammar provides an efficient way to parse the free word order of Korean, while the unification-based features express complex grammatical facts without complicating the parsing algorithm and, as a result, the parser can be easily modified for grammar correction, implementation of probabilities for the grammar rules, and application to other languages.

KorPar analyzes the structure of a Korean natural language sentence by representing it as a set of dependency pairs. Since Korean is a partially free-word-order language, KorPar accounts for restrictions on totally free order of the words in a sentence,

recognizes subcategorization features, restricts the order of dependents for a single head, matches long-distance dependencies, and parses nouns that lack case markers. KorPar has been tested with 100 consecutive sentences (more than 2000 words) from articles in the Chosun Ilbo Newspaper. The F-score (harmonic mean of precision and recall rates) was 96.3%.

INDEX WORDS: Dependency grammar, Unification-based grammar, Parsing, Natural language processing, Korean, Prolog

KORPAR: A RULE-BASED DEPENDENCY PARSER FOR KOREAN IMPLEMENTED IN PROLOG

by

SOYOUNG KWON

B.A., Ewha Womans University, Korea, 1996

M.A., Ewha Womans University, Korea, 1998

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2006

KORPAR: A RULE-BASED DEPENDENCY PARSER FOR KOREAN IMPLEMENTED IN PROLOG

by

SOYOUNG KWON

Major Professor: Michael A. Covington

Committee: Marlyse Baptista
Hyangsoon Yi

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2006

DEDICATION

To my parents, parents-in-law, my husband and my lovely two children who gave me love and support.

ACKNOWLEDGEMENTS

I would like to thank Dr. Covington for believing in me and giving me endless support throughout my courses, exams, and dissertation. I remember the first course that I have taken from you and how poorly I have written my paper in the class. I have got the worst grade among other courses, but I have learned the most valuable thing: to write only what you know and to express your knowledge with concise and clear sentences. I am so grateful that you have given me another chance and supported me during my graduate years. I will always cherish those years. Also, I would like to thank Dr. Baptista for always being there for me and supporting me like a sister. I have always admired your sincere warm heart and how you encourage us to pursuit our career. I hope I have made you proud and also wish the very best for you and your family.
I am also so grateful to Dr. Yi for all her encouragement and objective opinions and support in choosing my career. I have learned to plan my future with confidence thanks to you and hope that I could someday give in return.

Most of all, I would like to thank my family: my mom and dad who have sacrificed so much for my education, my mother-in-law who has always supported and encouraged me to finish my degree, my best friend and husband who I could not have made it without, and my lovely son and daughter who are the best children that any parent can ever have. I love you all so dearly!

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Natural language parsing is an integral part of natural language processing, since it is related to semantic interpretation, speech production, machine translation, information extraction, question answering, and so forth. Parsers based on various formalisms have been developed on both theoretical and statistical principles throughout the past decades. In recent years, statistical parsers have been widely used to offset the highly ambiguous property of language. However, the probabilistic parsers generally provide "the most likely" analysis, and in order to improve the accuracy of results, a large corpus of annotated sentences is required. As a result, the latest trend in parsing methods is to emphasize on a rule-based framework for accurate analysis and combine probability-based framework for a wide-coverage and efficient analysis.

The changes in the methods of parsing are also reflected in Korean parsers. In the 1980s, parsers based on various grammar formalisms, such as phrase structure rules, categorial grammar, and dependency grammar, were presented. Unfortunately, these parsers were not efficient and could not analyze a wide range of ambiguous sentences in Korean. To improve the efficiency and coverage of sentences, parsers based on statistics became popular in the 1990s. Nonetheless, language cannot be analyzed or understood using only the probability method because, in most cases, the parsing results are not accurate by simply providing the most likely analysis. Furthermore, a large annotated corpus that entails an enormous amount of time and effort is required to improve the accuracy of this method. Hence, nowadays, parsers based on grammar

rules are re-emphasized and furthermore, a probability framework added to these rules can improve the efficiency and accuracy of a parser.

1.1 STATEMENT OF THESIS

The parser, KorPar, presented in this dissertation, is a rule-based parser based on unification-based dependency grammar and is implemented in Prolog. Also, KorPar can provide a firm basis for implementing a hybrid parser by adding probabilities to the heuristic rules introduced in this work.

Korean is a partially free-word-order language. This means that most words in a sentence are free in order, though there are restrictions on certain words. For example, the verb and the attached postpositional tense marker and sentence-ending marker must be placed in the final position of a sentence. On the other hand, the noun and the attached postpositional case marker may be placed anywhere in a sentence, as long as they precede the verb. Dependency grammar is widely used to describe the word order and structure of a sentence in Korean parsers (Chung & Rim 2004; Kwon & Yoon 1991). However, many of the proposed parsers use probabilities (Lee 2002; Yoon 2002) and they lack a high accuracy rate, while others parse a sentence by taking the words in reverse order (Kim, Kim, Seo, & Kim 1994) and therefore the parser needs to wait for the entire words in the sentence to be input.

The hand-coded parser, KorPar, has been tested with 100 sentences (from articles of the Chosun Ilbo Newspaper) and showed a high performance rate of precision 97.6%, recall 95%, and the harmonic mean of precision and recall rate, F-score 96.3%. Although the number of test sentences is small, the sentences are composed of various dependency structures that seem to apply to almost all possible structures in Korean. If dependency grammar is combined with

unification-based grammar, the implementation of a Korean parser becomes simple and efficient. By representing a word with features and a grammar with the unification of features, the parsing algorithm is separated from the grammar allowing it to check syntactic, morphological, and semantic dependency. In addition, grammar correction, implementation of probabilites for the rules, and application to other languages can easily be achieved by modifying the lexicon and/or the grammar of the parser and leaving the algorithm itself unmodified. KorPar is implemented non-deterministically in Prolog using GULP (Graph Unification Logic Programming) (Covington 1994a), a tool for implementing unification-based grammar.

1.2. OVERVIEW OF DISSERTATION

This dissertation investigates the rule-based Korean parser KorPar. It analyzes its accuracy and efficiency and examines a number of potentially difficult troubleshooting cases and KorPar's ability to handle them.

Chapter 2 lists previous work related to parsing Korean. The proposed parsers based on phrase structure rules, categorial grammar, lexicalized tree-adjoining grammar, dependency grammar, and unification-based grammar are summarized. The proposed parsers based on probability are then reviewed, followed by a final section that briefly introduces the latest hybrid parsers of Korean, which have re-emphasized the rule-based framework.

Chapter 3 elaborates the characteristics of an ideal parser that has high accuracy, efficiency, and generative power. The probabilistic framework is questioned in regards to accuracy, the dependency grammar formalism is emphasized in regards to efficiency, and the rule-based framework is supported for its generative power. The final section explains how KorPar meets

all three conditions of an ideal parser, and provides support for the potential psychological realism of KorPar.

Chapter 4 introduces fundamental knowledge related to this dissertation. Dependency grammar, unification-based grammar, and the characteristics of the Korean language are briefly explained.

Chapter 5 describes how the parser KorPar functions. The lexicon, dependency grammar, and algorithm itself are explained in detail. In addition, the difficult cases for parsing Korean are presented and solutions are provided using examples. The cases include subcategorization, long-distance dependency, serial verb construction, adjectives and relative clauses, optional case marker, and supplementation of the hand-coded rules with probability.

Chapter 6 presents the results and evaluations of testing 100 sentences with KorPar and **Chapter 7** concludes the dissertation by discussing its contribution and possible future improvements.

CHAPTER 2

RELATED WORK ON PARSERS FOCUSING ON KOREAN

This chapter provides a summarization of previous work on parsing natural languages, with a focus on Korean. Research on parsing Korean sentences began in the 1980s, mostly with rule-based frameworks. Several different grammar formalisms were used in the rules of the parsers. A number of these early parsers faced various problems, and some eventually developed into a form similar to dependency grammar. In the mid 1990s, researchers tried to overcome these problems by using probabilities in their parsers. More recently, because of the difficulties in maintaining a large treebank and the limit in performing high accuracy rates by using only probabilities, the latest trend in parsing re-emphasizes a rule-based framework, which can be combined with a statistical framework.

2.1. RULE-BASED FRAMEWORK

2.1.1. PHRASE STRUCTURE GRAMMAR

Lee, Kim, and Kim (1997) proposed a variation of the phrase structure rule with restricted syntax. Since conventional phrase structure grammars are strict in word order, they needed a flexible grammar to accept the free word order of Korean. Eventually, the grammar appeared similar to dependency grammar:

> We developed a variation of phrase structure grammar with restricted

> syntax in this paper. The restriction aims at reducing the combinatorial complexity of rule numbers due to order sensitivity of phrase structure grammar. The restricted form of phrase structure grammar proposed in this paper resembles a dependency grammar. (Lee et al. 1997: 2)

For example, a sentence with a transitive verb in Korean can have either the subject preceding the object or vice versa. Therefore, the conventional phrase structure rule needed to be modified as (1), in order to parse the free word order of the subject and object.

(1) a. conventional non-restricted rule
VP → NP + object marker NP + subject marker VP

b. modified restricted rule
VP → NP + object marker VP
VP → NP + subject marker VP

According to Lee et al.'s grammar, the highest node in a sentence tree structure is VP and by modifying the conventional phrase structure rule to the restricted rule, it accepts free-word-order of subject and object in a sentence. The term "restricted" is used because the rule allows only one functional word (in this case, *object/subject marker*) in the right hand side of the rule. Eventually, the restricted phrase structure grammar is similar to dependency grammar by setting a modifier-modifiee relation between two words (phrases).

The proposed parser based on restricted phrase structure grammar showed almost a 50% decrease in the number of parsed trees per sentence and also less parsing time compared to the conventional non-restricted grammar.

2.1.2. CATEGORIAL GRAMMAR

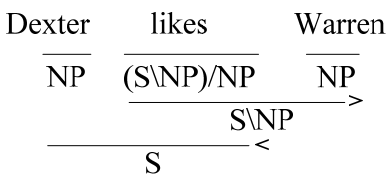
Categorial grammar is a form of lexicalized grammar in which the syntactic rules are expressed by *categories*, either *primitive categories* or *functions*. Primitive categories are generally, N, NP, PP, S and functions are normally VP.

Categories can be combined by two general function application rules shown in (2).

(3) a. $X / Y \quad Y \rightarrow X$
b. $Y \quad X \backslash Y \rightarrow X$

For example, a transitive verb has the notation of (S\NP)/NP as (3) (Steedman 1993).

(3)



Therefore, analogously to cancellation in the mathematical fractions, the verb *likes* combine with the object *Warren* and results in S\NP and in turn, the subject *Dexter* combines with S\NP and results in S.

Lee and Lee (1995) parsed Korean using categorial grammar, emphasizing the morphological structure of the agglutinative language. In addition, they developed the parser to parse spoken Korean by integrating phoneme-level inputs into morpheme-level inputs. However, the result of morphological analysis was perfect while the syntactic analysis was poor with

35.5% accuracy. Lee, Lee, and Lee (1995) further showed improvement of the parser to 61.8% accuracy in syntactic analysis, by using *n*-best strategy along with the semantic processing.

Cho and Park (2000) furthered research in this area by using categorial grammar and semantic knowledge to parse coordinate constructions. Since categorial grammar cannot represent and use lexical information, such as meaning, tense, number, etc., they combined the lexical semantic information to their system, which seemed similar to unification-based grammar.

2.1.3. LEXICALIZED TREE-ADJOINING GRAMMAR

Tree-adjoining grammar consists of a set of *elementary* trees: *initial* and *auxiliary* trees. These trees constitute the basic building blocks of the formalism and have the advantage of stating dependency relations between nodes of trees which are further apart. The *adjunction* and *substitution* operations build *derived* trees from elementary trees. (Noord 1993)

Han, Yoon, Kim, and Kim (2000) developed a wide-coverage Korean grammar based on lexicalized tree-adjoining grammar. This grammar analyzes inflectional morphemes, derivational morphemes, and the subcategorization of verbs. In addition, the adjoining trees are specified with features and the features are unified, allowing local constraints in subcategorizations to be processed.

2.1.4. DEPENDENCY GRAMMAR

Kwon and Yoon (1991) implemented a parser based on dependency grammar and implemented in C language to parse free-word-order sentences in Korean. Here, each morpheme is represented in feature structures and unification is performed on a head and dependent pair.

They proposed an approach that could analyze languages that are free in word order and have ellipsis, but did not provide any numerical evaluations.

Nam, So, Kim and Kwon (1998) presented a Korean grammar checker for simple sentences (without subordinate clauses or coordination) based on heuristic rules only.

Kim, Kim, Seo, and Kim (1994) proposed a dependency parser that processes the words in reverse order, from the last word of the input sentence to the first word. By processing an input sentence in reverse order, the number of possible dependents decreases since Korean is a head-final language and in turn, it increases the parsing speed. However, this approach lacks real-time application as it requires the parser to wait for the input of the entire sentence.

Germann (1999) presented a deterministic dependency parser for Japanese implemented in C++. This parser also parses in reverse order based on heuristic hand-coded rules.

2.1.5. UNIFICATION-BASED GRAMMAR

Unification-based grammar can be used to implement and extend various grammar theories. Kwon, Yoon, and Kim (1990) presentend KORANS (KORean ANalysis System), a parser based on Head-Driven Phrase Structure Grammar (HPSG). The rules based on HPSG is expressed in binary form as (4).

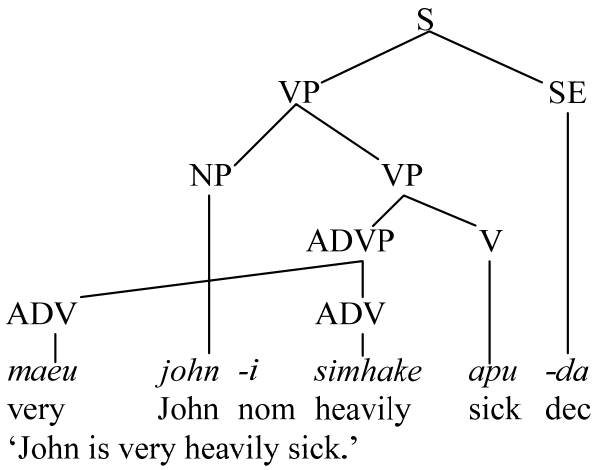
(4) M → D H
M → H
(M: Mother Category, D: Daughter Category, H: Head)
(from Kwon et al 1990:252)

However, the crossing of branches in Korean phrase structures is not allowed in (4). Thus, in addition to the HPSG rules, they used heuristic rules based on the characteristics of Korean to

link discontinuous constituents, which is a similar approach to Kim et al.'s modified phrase structure grammar in Section 2.1.1.

Kwon et al. (1990) further explained the heuristic rules with the sentence in (5).

(5)



(SE: Sentence Ender)

The first adverb *maeu* cannot be linked with either the second adverb *simhake* nor the verb *apu*, based on the HPSG rules. So the heuristic rules based on Korean require the first adverb to be linked with the second adverb, allowing a cross branching of the tree structure.

As mentioned in Section 2.1.4, Kwon and Yoon (1991) described a unification-based dependency parsing method implemented in the C language.

Agel, Eichinger, Eroms, Hellwig, Heringer, and Lobin (2003) introduced Dependency Unification Grammar (DUG), a linguistic programming language. Since pure dependency grammar cannot represent all linguistic structures, DUG formalism uses two additional operations: feature unification and tree transformation. Feature unification provides a detailed

representation of a sentence and tree transformation simulates all operations on a given sentence to form new ones such as paraphrasing, translating, summarizing, and so on.

2.2. PROBABILITY FRAMEWORK

Statistical approaches for natural language processing became popular in the mid 1990s (Charniak 1997; Samuelsson 2000). Korean parsing was no exception to this trend.

Yoon (2002) presented a parser that is controlled not by hand-coded heuristic rules, but by the statistical information extracted from the corpus. The dependency relation between words is based on the probability of a lexical association between the words. The lexical association is measured using linguistic knowledge and distance information from the corpus and the two words that has the highest association are linked together. The precision rate was 84.8% and the longer the sentence, the lower the precision rate.

Chung and Rim (2004) applied a similar approach by using probabilities of distance between words. Although the performance of the parser decreased 3.6%, it offered the advantage of using less time and space. They argued that the parser performs better once it considers additional contextual information.

2.3. HYBRID FRAMEWORK

From the late 1990s, the rule-based framework was re-emphasized due to the accuracy problem of the probability framework. Therefore, hybrid frameworks of rules and the probabilities of the rules have been proposed.

Cha, Lee, and Lee (2002) combined categorial grammar with statistics to parse sentences with coordination, long-distance scrambling, double subjects, and double objects. In a similar

approach, Clark, Hockenmaier, and Steedman (2002) presented a statistical English parser that uses combinatory categorial grammar to derive dependency structures, focusing on long-distance dependencies in coordinations, extractions, and raising verbs.

Kim, Kang, and Lee (2001) used valency information and the structural preference rule along with statistical information from a corpus for resolving ambiguities in dependency parsing. Lee & Choi (1997) also presented two kinds of algorithms based on probabilistic dependency grammar: reestimation and best-first parsing. Both algorithms showed $O(n^3)$ time complexities. Schneider (2003) introduced a hybrid dependency parser with low complexity for English. The probabilities are based on the distance between the head and dependent and the categories of words involved. Schneider pointed out the need for larger dependency-annotated corpora, since English has low counts of long-distance dependencies.

Considering various different approaches to parse either a relatively free-word-order language (such as Korean) or relatively fixed-word-order language (such as English), I propose a rule-based parser based on dependency grammar for efficiencies in representing the structure of partially free-word-order Korean sentences and unification-based grammar to identify head and dependent relations in a simple manner. Although there are rule-based parsers for Korean presented in the past, some lacked efficiency and some did not provide performance evaluations. The rule-based dependency parser, KorPar, is efficient and shows high accuracy rate and is implemented in Prolog for unification and backtracking. Also, it can be used as a basis for implementing probability parsers by using the grammar rules in the parser. The next chapter discusses the characteristics and advantages of KorPar compared to the previous proposed parsers by explaining the technical background of the parser.

CHAPTER 3

TECHNICAL BACKGROUND OF KORPAR

3.1. PROBABILITY AND ACCURACY

Parsers with a probability framework heavily depend on treebanks and morphological taggers. According to Zeman (2002), a parser's accuracy can decrease depending on the treebank design and the errors of the morphological tagger. Zeman argues that building a treebank and/or morphological tagger requires a complicated and expensive effort and it may not always have a positive effect on the performance of the parser. It is apparent that the extraction of information from the corpora needs to be facilitated by human linguistic knowledge: hand-coded rules.

Rayner, Boullion, Hockey, Chatzichrisafis, and Starlander (2004) compared two versions of a speech translation system, a grammar-based language model and a statistical language model. Both systems were carried out on the same corpus and evaluation was based on the performance of translation. They concluded that the grammar-based language model system outperformed the statistical one due to its ability to present more predictable interface to the user.

Therefore, the ideal framework in natural language processing is to create a broad-coverage, rule-based parser, using additional statistical methods to adapt the rule-based system automatically and to acquire information from the corpora (Richardson 1994). The hybrid system has practical application because creating a large corpus database for a probability framework, is expensive, time consuming, and lacks highly accurate performance.

3.2. DEPENDENCY GRAMMAR AND EFFICIENCY

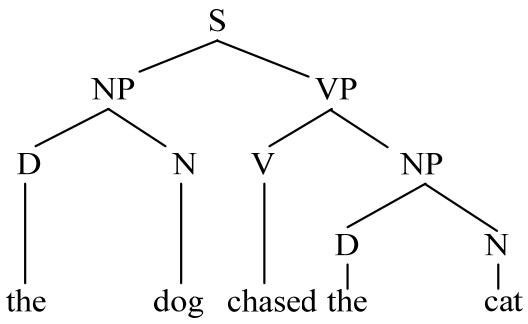
A parser provides a syntactic structure of unrestricted natural language based on certain syntactic representations such as constituency grammar or dependency grammar. KorPar presents structures of input sentences based on dependency grammar. Compared to constituency grammar, dependency grammar is a suitable syntactic representation for parsers of relatively free-word-order languages.

3.2.1. DEPENDENCY GRAMMAR VS CONSTITUENCY GRAMMAR

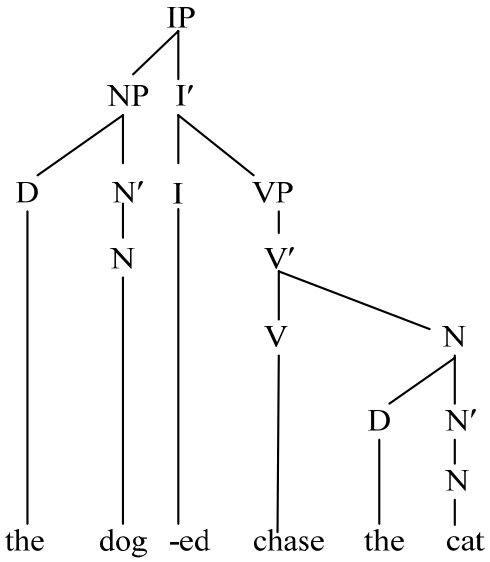
There are two different types of structural descriptions widely used, constituency grammar and dependency grammar.

Constituency grammar (CG) represents a sentence as a certain constituent (phrase), which in turn consists of smaller constituents (phrases) or words. Therefore, CG groups words into constituents. For example, *The dog chased the cat* is represented as:

(6) a.



b.

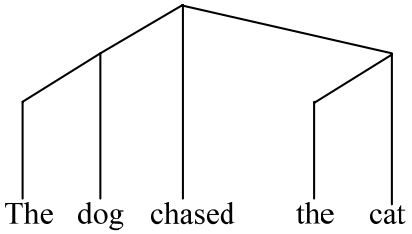


The structure in (6a) represents the sentence based on phrase structure rules, whereas (6b) is based on the X-bar theory. Both representations group *the cat* to generate a noun phrase, while *chased* is added to generate a verb phrase and *the dog* is added to complete the whole sentence. According to Jackendoff (1977), the labels of the nodes in (6a) have no theoretical explanations as to why N is related to NP and V is related to VP. In particular, where did the node S come from? The labels NP or VP are just symbols, unrelated to N and V respectively, and could have been labeled as X or Y. In comparison, the labels of the nodes in (6b) do have explanations. The central claim of X-bar theory is that the labels N, N´, and NP are related, and this is known as the "endocentric constraint." This constraint states that all constituents (phrases) must have a head and this head projects to a higher node. V, V´, and VP are related in the same way, as are P, P´, and PP.

Dependency grammar (DG) links two words at a time, one being the head and the other being the dependent. Normally in a sentence, all but one word depend on another word. The

one word that does not depend on another word is the ultimate head of a sentence.[1] There are several different criteria for deciding head and dependent pairs, which will be discussed in detail in 4.1. The representation of the sentence above is shown in (7).

(7)



The dependency pairs are graphically represented by a downhill line from the head to the dependent. In DG formalism, the actual word is linked with another word to create a dependency tree.

The difference between CG and DG is that DG does not have higher nodes (intermediate levels). The actual words are linked together, not the abstract nodes. Although there are no intermediate levels in DG, the notion of constituency does exist. Covington (1992) describes a constituent in DG as a head and its dependents, their dependents, and so on, recursively. The only difference is that DG does not emphasize the notion of constituency, but rather the relation between a head and its dependent(s).

[1] Hudson (1984) argues that sentences with raising verbs have multiple heads. For example, *John* in *John seems to like Mary* has two heads: *seems* and *like*.

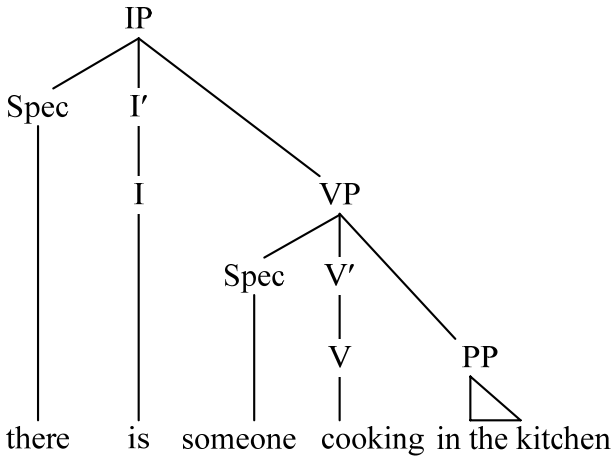
3.2.2. CONSTITUENCY GRAMMAR AS DEPENDENCY GRAMMAR

Recent developments in the formalism of CG have made it much more like DG. First, the change from the phrase structure rules to the X-bar theory shows that CG is becoming similar to DG. The difference between the two, as mentioned in 3.2.1, is the endocentric constraint. In the X-bar theory, every constituent (phrase or sentence) should have a head, but in phrase structure grammar, there is no head for a sentence. The S node in (6a) is not a projection of any other nodes in the tree and therefore, the sentence lacks a head. However, in the X-bar tree, in general, V is the head of VP, N is the head of NP, and I is the head of IP (which is similar to S node in phrase structure rules), and the constituents in each phrase other than the head are its dependents. In X-bar theory, the ultimate head of a sentence is I (the inflection of the verb in a sentence) and the subject occupies the Spec of I. Like DG, which analyzes the verb in a sentence to be the ultimate head[2] and the subject to be the dependent of the verb, CG started to show supremacy of verbs.

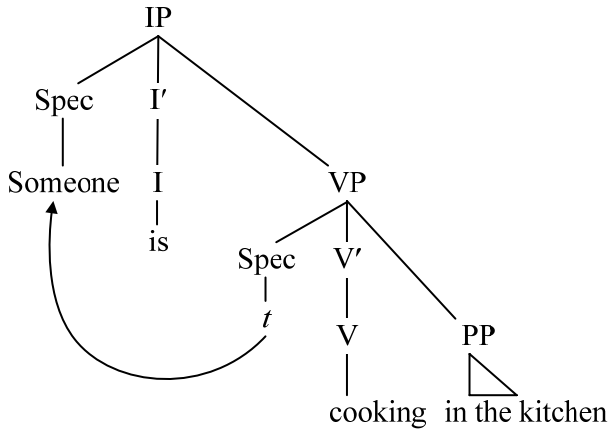
Second, the "VP-internal hypothesis" emphasizes verbs also. This hypothesis of the Government and Binding Theory claims that the subject initially occupies the Spec position of VP (Webelhuth 1995). Sentences with expletives can support this hypothesis as in (8).

[2] In general, the ultimate head in English sentences is analyzed to be the verb. In comparison, KorPar analyzes the postpositional sentence-ending marker that attaches to the verb to be the ulimate head.

(8) a.

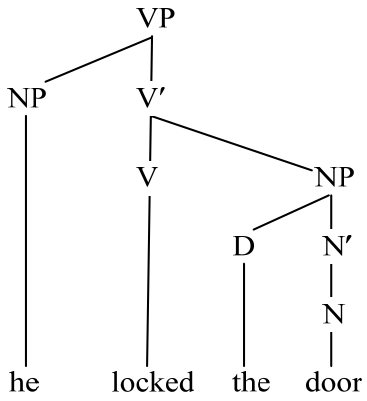


b.

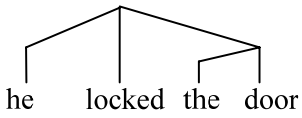


Two sentences with the same meaning can be expressed either with or without an expletive *there*. If it is expressed with the expletive *there*, *someone* occupies the Spec position of VP and *there* occupies the Spec of IP (8a). If the same meaning is expressed without the expletive *there*, we can assume that *someone* in the Spec of VP in deep structure has moved to the Spec position of IP in surface structure (8b). Again, the subject depends on the verb and this hypothesis shows the supremacy of verbs, similar to DG. Therefore, the representation of the sentence *He locked the door* based on CG and DG becomes similar when following the VP-internal hypothesis.

(9) a.

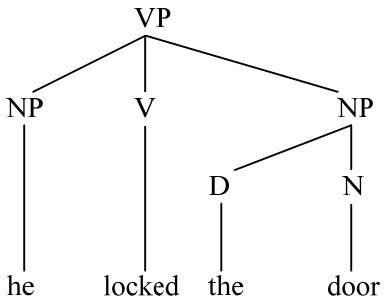


b.



If there were only one non-terminal level in (9a), the tree structures based on the X-bar theory (9a) and dependency grammar (9b) would become even more identical as in (10).

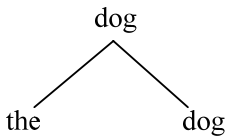
(10)



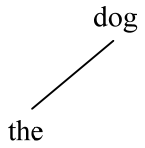
Third, Chomsky's Minimalist Program (1995) questions the necessity of the intermediate bar-levels. He argues that only the head and the maximal projection relate to the interpretation

of the logical form (LF) and the phonetic form (PF) interface. The maximal projection can be labeled using the label of the head, and labeling the higher nodes can be eliminated since the feature information is included in the head. Therefore, according to the Economy Principle,[3] the intermediate levels should be abandoned and labeling higher nodes should be considered redundant. This results in the bare phrase structure. Schneider (1988) showed the representation for the noun phrase, *the dog,* based on the bare phrase structure (11a) and DG (11b). (11a) shows the process of 'merge'[4] in bare phrase structure: *the* and *dog* merge together and the head of the phrase is *dog*. (11b) shows the head *dog* and the dependent *the* by drawing a downhill line from head to dependent.

(11) a.



b.



[3] A principle which requires that (all other things being equal) syntactic representations should contain as few constituents and syntactic derivations involve as few grammatical operations as possible (Radford 1997).

[4] The operation that combines lexical items or partial trees with others is called 'merge'. Merge is always a binary relation, that forms larger units out of those already constructed (Schneider 1988).

Accordingly, examples (11a) and (11b) appear the same, except for the repetition of the head in (11a).

Kornai and Pullum (1990) also argued that the X-bar theory has developed with too much emphasis on the bar levels. They contended that it is more important to establish what the head of the constituent in question is, than to label the higher nodes. In other words, the "headedness" should be emphasized for the descriptive power of representing languages.

Therefore, the X-bar theory introduced the notion of "headedness" in CG, and since then, CG has become similar to DG.

3.2.3. WHY IMPLEMENT DEPENDENCY GRAMMAR IN KORPAR?

There are two main advantages of dependency grammar. One advantage is that DG has fewer nodes than CG**.** There are no abstract higher nodes, no empty nodes, no traces, and no transformations (movements). In the practical field of parsing, the grammar should be as simple as possible for ease of implementation of the algorithm. If the parser needs to simply link the actual words, not analyze additional abstract units, then the task becomes easy and straightforward. In addition, parsing based on DG is achieved one word at a time, while CG must wait for constituents to be formed.

However, when we consider the complexity of the parsing task, it could be the case that parsers that use CG or DG formalism are NP-complete. According to Barton, Berwick, and Ristad (1987), if agreement features are considered in the grammar, the complexity becomes exponential. We need to take into account that it is rare for a natural language to have ridiculously long sentences that typify the worst-case complexity of a parser (Covington 1994b:193-194; Voll, Yeh, and Dahl 2001).

Another advantage of DG is its ability to represent the free word order of languages in a simple formalism. Since DG links two words at a time, word order is not emphasized as in CG. Moreover, if "non-projectivity" is allowed, the grammar can accept totally free-word-order languages. Non-projectivity means that the arcs in the dependency tree can be crossed. (See (71) in Section 5.4.2.) In other words, DG can represent discontinuous constituents, which often appear in free-word-order languages.

3.3. RULE-BASED PARSER AND GENENRATIVE POWER

As mentioned in Section 3.1, natural language processing, and specifically natural language parsing, should be based on linguistic knowledge to achieve a better performance in accuracy. It is difficult to implement a parser based solely on probability. The parser requires rules that explain how an input sentence is grammatical and that provide the internal structure of the sentence (Jäarvinen & Tapanainen 1998). In other words, a parser needs rules that are generative. A generative grammar must consist of a set of rules that define classes of well-formed sentences. These sets of rules assign a structural description to each well-formed sentence that provides a basis for explaining native speakers' judgments about pronunciation, meaning, structure, and structural relations. We call a grammar with these rules a descriptively adequate grammar.

3.4. SATISFYING 3.1, 3.2 AND 3.3 WITH KORPAR

3.4.1. RULE-BASED DEPENDENCY GRAMMAR

KorPar uses a rule-based framework of unification-based dependency grammar. The grammar rules are hand-coded based on dependency grammar to accept the free order of the words in a sentence and the unification-based grammar simplifies the task of identifying the head and dependent relations. Therefore, it shows high accuracy performance and has generative power because of the heurisitic rules in the grammar, and also shows efficiency in implementation since it is based on dependency grammar.

3.4.2. PSYCHOLOGICAL BACKGROUND TO SUPPORT KORPAR

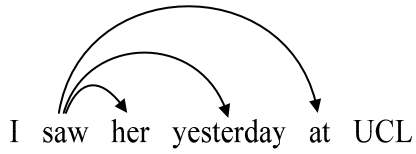
Jackendoff (2002) provided a perspective on how language is perceived and produced by re-evaluating the concept of generative grammar introduced from Chomsky's *Aspects of the Theory of Syntax* in 1965 to the Minimalist Program. According to the mainstream generative grammar, human can produce and perceive unlimited number of new utterances and also, utternces with great length and complexity. For example, sentences such as, *We have a young child and our chef has twins so we know how difficult it is to find a first rate restaurant that doesn't shudder when you show up at the door with kids in tow* (from Jackendoff 2002:39), could be understood and produced by the combinatoriality of finite list of lexical items and finite set of grammar. Jackendoff does agree with this concept, but he points out that it is mistakenly *syntactocentric*: syntax is the only generative component and it drives both phonology and semantics. Instead, Jackendoff proposed a parallel constraint-based architecture, which views syntax, phonology, and semantics as equally independent, generative and combinatorial levels of structures. Also, each level of structure is characterized by its own set of primitives and combinatorial principles (rules) and these three combinatorial systems are connected by interface constraints, which are the words in the lexicon. The parallel constraint-based architecture is also

non-directional: one can start with any piece of structure in any component and pass along the information to any other component.

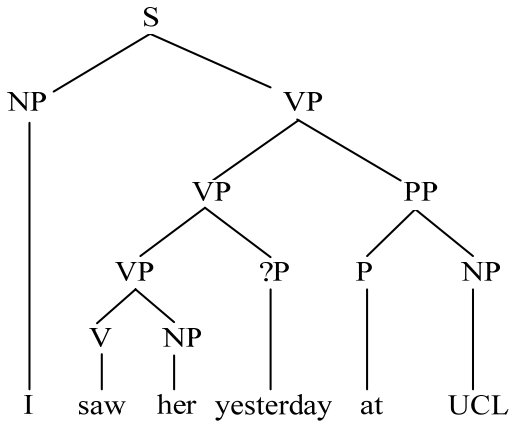
Accordingly, I believe that human perceive and produce unlimited new utterances based on a limited set of lexicon and grammars, not based on probability. We do not guess the most likely meaning and structure of a sentence, instead we parse a sentence with accuracy. Therefore, a rule-based parser should be emphasized for its psychological plausibility. Also, the order in which components are activated is not determined and KorPar follows a similar process by unifying features order-independently.

Hudson (1993) supported dependency grammar by arguing that the higher nodes in a tree structure are psychologically implausible. He suggested that the hearer's task is to establish a syntactic and semantic relations between each word and its head. Therefore, building higher nodes is irrelevant to this task. For example, in processing the sentence, *I saw her yesterday at UCL,* the DG formalism is flat and simple as in (12a). Each adjunct *yesterday* and *at UCL* is directly linked with the common head *saw*.

(12) a.



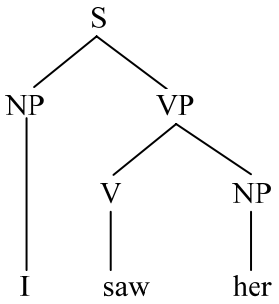
b.



(from Hudson 1993:284)

However, based on the CG formalism (more accurately, the Extended Standard Theory) as in (12b), the relations between V and S changes from being separated by one VP to eventually three VPs as each adjunct is added to *I saw her*. (Compare (12b) with (13).)

(13)



These changes in the tree structure should cause complexity of processing since each adjunct introduces different structural relations between the verb and the sentence. But in reality, the sentence is not problematic for a hearer and the hearer does not backtrack to set up a new relation

between V and S. Therefore, the higher nodes VP that introduce the adjuncts are theoretically odd.

A parser based on dependency grammar and Prolog can also be supported by Abney's (1989) analysis of human parsing. According to Abney, "Human parsing is deterministic in most inputs, but if parsing is done non-deterministically, it backtracks. The human parser does not pursue multiple parsing in parallel and it does not wait until the whole phrase (constituent) is composed."

Therefore, an artificial parser should be able to analyze a sentence as similar as possible to the human parser and at the same time, the implementation should be efficient in time and space. In order to do so, we need to understand the psychological background of the human parser and choose the appropriate formalism to represent the structure of a given natural language input. As we have seen above, there are proposals that the human parser analyzes languages based on some sets of grammars and therefore, I strongly believe that a rule-based parser needs to be emphasized. Also, for efficiency in implementation, the DG formalism is appropriate because it has fewer nodes (the actual words in a sentence) than CG formalism and it easily represents free-word-order languages.

The next chapter will briefly introduce the criteria for identifying head and dependent relations, the software tool for implementing unification-based grammar, and the characteristics of Korean.

CHAPTER 4

THEORETICAL BASIS OF KORPAR

4.1. DEPENDENCY GRAMMAR

4.1.1. BASIC CONCEPT OF DEPENDENCY GRAMMAR

Dependency grammar describes sentence structures by linking individual words and specifying their relations (Bauer 1979, Hays 1964). Each link will have a **head** and a **dependent.** In general, the dependent is the modifier or complement and the head determines the attribute of the dependent. The head is obligatory and the dependent is optional in a sentence. Although there are variations in identifying head and dependent relations among languages, the ultimate head—a head that is not a dependent of any other head—would normally be the single head for a single sentence.

The head and dependent pair can be represented graphically in many different ways. For example, indentation can be used by placing the ultimate head in the leftmost position and indenting its dependents or by drawing a line downhill from head to dependent or pointing an arrow from head to dependent (Covington 1990).

4.1.2. CRITERIA FOR IDENTIFYING DEPENDENCY

The term “head” has been used in the field of linguistics regardless of the grammatical theory described. However, there is no single explicit notion of “head” that all linguistic theories

agree upon. A head could have different meanings and functions in constituent grammar and dependency grammar. There are several criteria that KorPar has chosen to follow when identifying head and dependent pairs.

According to Mel'čuk (2003), a single head and dependent pair is linked based on three levels of dependencies: semantic, morphological, and syntactic dependencies. First, an argument of a predicate semantically depends on its predicate. For example, the sentence, *The boy gave his friend a present* can be analyzed such that each argument, *the boy*, *his friend*, and *a present,* semantically depends on the predicate *gave*. In other words, the arguments need the predicate to assign its semantic role in a sentence either as an AGENT, THEME, RECIPIENT, etc. Second, if a word *w1* assumes a certain morphological form under the influence of another word *w2*, then *w1* morphologically depends on *w2*. For example, in the sentences, *He likes apples* and *They like apples*, the predicate *like* has different forms, depending on the subject argument. Third, the syntactic dependency is an intermediate between semantic and morphological dependencies. The dependency tree contains the meaning information (semantic) as well as the phonological form (morphological) information of the dependency pairs.

Similar to Mel'čuk, Kruijff (2002) proposed several characteristics of head and dependent pairs. First, the head is obligatory and the dependent is optional. Second, the dependent modifies the head, thus the pair becomes a hyponym of what the head alone refers to. Third, the syntagm of the pair remains in the same category as the syntactic syntagm of the head. Fourth, the head represents the external relation in a sentence. Fifth, the head decides the valency structure. Sixth, the head contains the inflections. Finally, the head decides the morphosyntactic form(s) of the dependent(s).

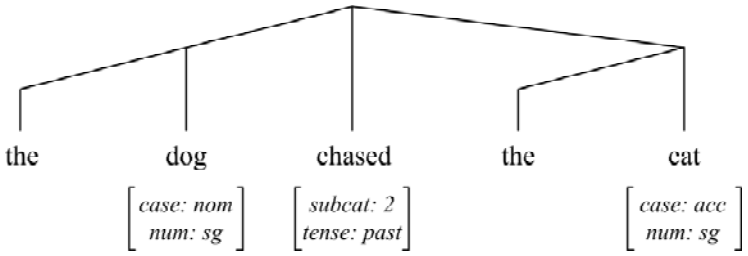
There can be controversies between the final two criteria because the head can either contain inflections and/or determine the inflections of the dependents. If we consider two sentences, *The child plays* and *The children play*, should the subject be the head or the verb be the head of these sentences? Kruiff explains these sentences as *mutual dependency* following Mel'čuk (1988). Mutual dependency is defined as, "The main verb and its grammatical subject are mutually dependent: the verb represents the clause but the subject controls the verb's form." Furthermore, the verb is the head of the subject based on syntactic dependency and the subject is the head of the verb based on morphological dependency. Therefore, dependency relations cannot be indentified satisfying only one criterion or satisfying all the criteria.

4.2. UNIFICATION-BASED GRAMMAR AND GULP

Unification-based grammar uses features and their values to represent grammar and features are grouped into feature structures (Shieber 1986). Each feature structure has information on meaning, part of speech, number, tense, case, subcategorizations, and so forth. In KorPar, only the features that are related to the word are instantiated; the others are left uninstantiated. These features and their values play a role in deciding syntactic, morphological, and semantic dependency. The values of the instantiated features in the feature structure are unified (matched or merged), order independently.

Unification-based grammar is a mechanism that can be used in various grammar theories. For example, dependency grammar can be implemented with feature structures, as (14a). The dependency relations are shown in downhill lines and the features and their values are represented in square brackets under the lexical item.

(14) a.



b. **chased (subcategorization:2..tense:past)**
dog(case:nom..num:sg)
the
cat(case:acc..num:sg)
the

The software tool, GULP (Graph Unification Logic Programming) (Covington 1994a), facilitates the implementation of unification-based feature structures in Prolog. The syntax of GULP is, for example, **a:b..c:d**, which represents a feature structure of **a** having the value **b** and **c** having the value **d**, and all other features are uninstantiated. The operator '**:**' joins a feature to its value and the operator '**..**' combines the feature and its value pairs to build more complex structures. The GULP feature structures are automatically translated to their internal Prolog terms when the program is compiled and then these structures are translated back when output by **print**.

The same sentence in (14a) can be represented in unification-based dependency grammar using GULP notations as in (14b). The features and their values are represented in parentheses, alongside the lexical item, and the indentation shows the dependency relations. Compared to (14a), the GULP notation is concise and simple for implementation.

4.3. CHARACTERISTICS OF KOREAN

There are three major characteristics of Korean that should to be addressed. The first characteristic is the postpositional function words. These are bound morphemes, each carrying different functions such as case marker, tense marker, and plural marker (Lee 2005).

The second characteristic of Korean is word order. Korean is a partially free-word-order language, such that most of the words in sentences are free in order. However, there are restrictions on some words:

- Verb and postpositional verbal ending(s) must assume the final position of a sentence, while the words before the verb are free or even omitted, as in (15a) and (15b).
- Postpositional function words must immediately follow the words they modify, as in (15c).
- Adverbs that modify verbs (in simple declarative sentences) may be placed in any position, as long as they precede the verb. In comparison, adverbs that modify adjectives must be placed immediately before the adjective, as in (15d).
- Adjectives and relative clauses must come right before the noun they modify, as in (15e).

These restrictions are illustrated with examples in (15).

(15) a. *gang-a-ji* *ga* *go-yang-i* *lul* *chot* *nun* *-da.*
dog nom cat acc chase pres dec
'The dog chases the cat.'

b. *go-yang-i* *lul* *gang-a-ji* *ga* *chot* *nun* *-da.*
cat acc dog nom chase pres dec
'The dog chases the cat.'

c. *go-yang-i* *gang-a-ji* *ga* *lul* *chot* *nun* *-da.*
cat dog nom acc chase pres dec
ungrammatical sentence

d. *nae* *ga* *aju* *ken* *cha* *lul* *po* *n* *-da*
I nom very big car acc see pres dec
'I see a very big car.'

e. *nae* *ga* *jaju* *po* *-n* *cha* *ga* *ga* *n* *-da*
I nom often see rel car nom go pres dec
'The car that I saw often, is going.'

The third characteristic of Korean is subcategorization. There are many cases of null arguments in Korean. Therefore, subjects and objects can be dropped and dependency grammar can easily represent these sentences. For example, if a verb is categorized as having two subcategories, it means that the verb can have, *at the most,* two subcategories. For example, sentences with transitive verbs can have only one argument realized.

(16) *gang-a-ji* *ga* *chot* *nun* *-da*
dog nom chase pres dec
'The dog chases _.'

These characteristics and also others will be further discussed in 5.4, concerning how KorPar manages to recognize and correctly analyze them.

CHAPTER 5

KORPAR (KOREAN PARSER)

KorPar is a rule-based parser for Korean based on dependency grammar and unification-based grammar. The dependency grammar can easily represent the structure of the free order of words in a sentence by linking two words at a time. The features of each word are unified in order to check syntactic, morphological, and semantic dependency for a single head and dependent pair. The unifications of features are done order independently in the grammar, and thus the grammar is separated from the algorithm itself.

KorPar has three sections: the lexicon, the dependency grammar, and the parsing algorithm itself. It is implemented in Prolog and the choice of lexical entries, dependency rules, and predicates in the algorithm introduce nondeterminism.[5] (See Appendix for the Prolog program listing of KorPar.)

5.1. LEXICON

The lexicon is a list of words represented in feature structures, as shown in (17).

(17) **word (PhoneticSounds, PartofSpeech(GULPFeatures)).**

[5] Harri & Oy (1998) introduced a dependency parsing algorithm that parses Finnish sentences deterministically in linear time. Also, Germann (1999) presented a rule-based, deterministic dependency parser for Japanese.

PhoneticSounds is the actual input word from the sentence to be parsed. Categories of **PartofSpeech** are shown below in Table 1.

Table 1. Categories of **PartofSpeech**

PartofSpeech	Symbol	Example
noun	n	gang-a-ji (‘dog’), yeungu (‘research’)
postpositional case marker	pp	ga (nom/vow/sing), i (nom/sing)
postpositional plural marker	plp	dul (plural)
adverbial post marker	advp	euro (‘to’, ‘as’), bute (‘since’)
verb	v	chot (‘chase’), malha (‘say’)
sentence-ending marker	p	da (declarative)
tense post marker	tp	nun (pres/con), go-iss (pres-prog)
copular verb	cp	-i (‘be’)
other kind of copular verb	ccp	dwe (‘become’)
serial verb marker	serv	-e
adjective	adj	i-ben (‘this time’), jen (‘previous’)
genitive post marker	gen	euy (‘of’), la-nun (‘so-called’)
numeral	num	myut (‘several’), du (‘two’)
c 1	ncv	la-go (‘says-that’), wihaese (‘for’)
c 2	scs	ha-go (‘and’), i-na (‘or’)
c 3	pcv	go (‘that’), hae-do (‘although’)
c 4	vcv	nun-ji (‘if’), l-surok (‘the-more’)
c 5	vcn	-nun (pres), -n (past/vow)
adverb	adv	sil-jae-ro (‘in reality’), jemjem (‘gradually’)
adverbs with tense	tadv	ap-euro (‘in the future’)
verb to noun	vnp	gi, eum
noun to noun	nnp	wha
quotation mark	qm	‘…’ / ‘…’

Different Korean linguists and lexicographers use different numers of categories of words in Korean. Choi (1971) presented ten categories: noun, pronoun, numeral, verb, adjective, determiner, adverb, particle, copula, and interjection. Sohn (1999), on the other hand, presented

eight categories: noun, pronoun, numeral, verb, adjective, determiner, adverb, and particle. These categories are based on each word's form, meaning, and function in a sentence.

KorPar categorizes words into categories similar to Choi's (1971) and Sohn's (1999), though there are a few differences. Since the main goal of KorPar is to grammatically check an input sentence and provide a syntactic dependency structure of the sentence, the categories of **PartofSpeech** are based on the morphosyntactic dependency relations, and the semantic properties are shown with the **sem** feature, which is one of the **GULPFeatures.**

Proper nouns and pronouns are categorized as nouns, since their syntactic relations and functions are similar to those of nouns. There can also be consecutive nouns that make compound words without any "connector." (See also 5.4.3.)

Adjectives and determiners are categorized based on a different analysis. KorPar analyzes adjectives as predicates that introduce relative clauses with the complementizer **vcn** and determiners as "pure" modifiers of nouns, labeled as **adj**. (See also 5.4.5.)

The particles are categorized into several different parts of speech based on their syntactic functions. Sohn (1999) subcategorized the particles into case, delimiter, and conjunctive cases and Lee (2003) subcategorized the particles as nominative, accusative, locative1, locative2, and instrumental. In comparison, KorPar categorizes the postpositional particles as case marker, plural marker, adverbial marker, tense marker, sentence-ending marker, serial verb marker, genitive marker, verb to noun marker, noun to noun marker, and complementizer/conjunction. The words that have similar meanings and functions to conjunctions/complementizers in English are categorized as **c**. Based on their dependency relations, **c** is subcategorized into **ncv, scs, pcv, vcv,** and **vcn**. For example, the postpositional complementizer **pcv** attaches to a sentence ending marker **p** and is followed by a verb, whereas **scs** is the kind of postpositional conjunction that

attaches to and is followed by the same part of speech (**n, v**, or **cp**). All five categories of the complementizer/ conjunction are shown in (18).

(18) a. *gamki* ***-edo-bulguhago*** *hakkyo* *- ey* *nao* *- ss* *-da*
cold **although (ncv)** school to come past dec
'Although _ had a cold, _ came to school.'

b. *hakkyo* *-ey* *ga* ***-gena*** *whesa* *-ey* *ga* *-n* *-da*
school to go **or (scs)** office to go pres dec
'Either _ go to school or to the office.'

c. *ge-ga* *hakkyo* *-ey* *ga* *-n* *-da* ***-go*** *malha* *-n* *-da*
henom school to go pres dec **that (pcv)** say pres dec
'He says that he goes to school.'

d. *hakkyo* *-ey* *ga* ***-l-su-rok*** *bae-u* ***-nun*** *geut* *-i* *man* *-da.*
school to go t**he more (vcv)** learn **(vcn)** thing nom many dec
'The more _ go to school, the more _ learn.'

The **scs** in (18b) uses the preceding verb *ga* as the dependent and the following verb *ga* as the head.

GULPFeatures are features notated in GULP syntax. These features are used in the unification framework to link the head and dependent pairs later in the grammar section of KorPar. There are seven features in the lexicon. The **sem** feature is the meaning of the word. The **tense** feature is the tense information of the **PartofSpeech tp**. The **num** feature is the number, either singular or plural. The **case** feature is either the nominative or accusative case for the nouns and the **subcat** feature is the subcategorization feature for the verbs. If the value is 1, it means that the verb is an intransitive verb. If the value is 2, it means that the verb is a transitive verb. The **ending** feature is a sound agreement between the head and the dependent. If the end of the preceding dependent is a vowel, then the following head must begin with a

consonant, and vice versa. The values **vow** and **con** are decided by the value of the dependent. For example, the noun *gang-a-ji* is a dependent that has the value **vow** and the postpositional case marker *ga* is the matching head that has the same value. The **feat** feature helps differentiate dynamic verbs and stative verbs since they have a different syntactic dependency relation, used either attributively or predicatively. (See also 5.4.5.)

Several examples of words in the lexicon are shown in (19).

(19) **word(gyosoo,n(ending:vow..sem:professor)).**
word(saebyuk,n(ending:con..sem:dawn)).
word(ga,pp(case:nom..ending:vow..num:sing)).
word(un,pp(case:nom-topicalization..ending:con..num:sing)).
word(euro,advp(ending:con..sem:to-or-as-or-with)).
word(jugo-bad,v(ending:con..subcat:2..sem:exchange..feat:dyn)).
word(ul-geut-i,tp(tense:future..ending:con)).
word(myun-se,vcv(ending:vow..sem:as-or-while)).

5.2. GRAMMAR RULES

5.2.1. ORDER RESTRICTIONS

As mentioned in 4.3, Korean is a head-final language and the dependent is always placed before the head. The head can be either adjacent or not adjacent to the dependent, depending on the pair. For example, an adjective and an adverb (modifying an adjective) must immediately precede the word they modify. On the other hand, the postpositional case markers should precede (but do not have to be adjacent to) the verb.

In order to implement these differences in word order restrictions, once a word enters the parsing algorithm, the notation of the word changes to a list **Node**, as in (20).

(20) **[Number, ListofDependents, PhoneticSound,PartofSpeech,GULPFeatures].**

Number is the position of the word in a sentence. **ListofDependents** is the list of all the dependent(s), each dependent's dependent(s), and so on. The **Number** atom is used in the **Conditions** of the dependency rules to state the order restriction (see (21)). For example, if the **Number** of the dependent is *N1* and that of the head is *N2*, the "adjacency restriction" states that *N2* is *N1* + 1, and the "preceding restriction" states that *N1* > *N2*.

5.2.2. HEAD AND DEPENDENT PAIRS

The dependency rules are represented in the form shown in (21).

(21) **check_dh(Dependent,Head) :- Condition.**

Two **Node** lists, **Dependent** and **Head**, are paired. **Condition** states the order restrictions (using **Number**) and the feature agreement restrictions (using **GULPFeatures**).

The head and dependent pairs are linked based on **PartofSpeech**. KorPar checks the parts of speech of the head and the dependent. Since Korean is a head-final language, the ultimate head is the sentence-ending marker **p**, which always appears in a sentence. The head and dependent pairs are listed in Table 2.

Table 2. Head and Dependent Pairs of KorPar

DEPENDENT	HEAD
v cp	p (ultimate head)
serv pcv vcv ncv pp adv advp	v
n advp gen	pp
adj nnp vnp	n
adv	adj

n	gen
n	nnp
v	vnp
v	serv
p (subordinate clause)	pcv
v	vcv
n	ncv

As mentioned in 4.1.2, there are several different criteria for identifying head and dependent. Consequently, there could be debates on the identification of head and dependent pairs in Korean. Kim, Kim, Seo, and Kim (1994) listed several different types of dependency relations, as presented in Table 3.

Table 3. Head and Dependent Relations as proposed by Kim, Kim, Seo, and Kim (1994)[6]

Governor	**Dependent**	**Dependency Relation**
Noun	noun, adnominal transformation ending	Modification
Noun	attributive and stative adverb	Addition
Numeral	pronoun, demonstrative adnoun	Modification
Numeral Adnoun	attributive and stative adnoun	Modification
Verb	noun, pronoun, numeral, case particle	Case Relation
…	…	…

According to Table 3, the head and dependent relations are based on a larger unit compared to KorPar. The head and dependent pairs listed in Table 2 are based on *eujel*, which is a basic unit

[6] The term "governor" is used instead of "head" in several Korean parsers. Also, Kim et. al did not provide the entire list of dependency relations.

that has either a meaning or a function. In comparison, the head and dependent relations in Table 3 are based on a word unit, which is a unit split by whitespaces. For example, in (18b), *hakkyo* and *ey* are individual eujels since each have either a meaning or a function (postpositional marker that indicates 'location'), while *hakkyo-ey* is the word unit separated from other word units by whitespaces in the writing system. Because Korean is an agglutinative language where several morphemes combine to form words, there are controversies in the definition and representation of heads and dependents (Kim, Choi & Lee 2003: 200).

The main goal of KorPar is to provide a dependency tree of the input sentence. As a result, the parts of speech are categorized according to this goal and the head and dependent pairs are stated based on semantic, morphological, and syntactic dependency.

Let us consider (22), for example.

(22) *na nun ecey san eyes kkweng ul cap - ass - eyo*
I nom yesterday mountain on pheasant acc catch past dec
'I caught a pheasant on the mountain yesterday.'
(from Sohn 1997: 293)

There are alternative word orders for this sentence.

(23) a. *na nun san eyes ecey kkweng ul cap - ass - eyo*

b. *na nun san eyes kkweng ul ecey cap - ass - eyo*

c. *ecey na nun kkweng ul san eyse cap - ass - eyo*

d. *na nun kkweng ul san eyse ecey cap - ass- eyo*

e. *kkweng ul na nun ecey san eyse cap - ass- eyo*

f. *san eyse ecey na nun kkweng ul cap - ass- eyo*

The sentences in (23) are a few (not all) of the alternative word orders for the sentence in (22). The verb and postpositional particles attached to the verb *cap - ass - eyo* are always placed in the final position of the sentences. However, the postpositional case marker and the adverbial marker that attach to nouns are free in order. Which word, then, should be the ultimate head, and how should we link the head and dependent pairs?

In 4.1.2, the criteria for identifying head and dependent were introduced. A brief summarization is as follows:

(1) The head is obligatory and the dependent is optional.

(2) The head and dependent pair is a hyponym of the head.

(3) The head determines the external relations and the valency structures.

(4) The head contains the inflections.

(5) The head determines the morphosyntactic form of the dependent.

There are several reasons why KorPar analyzes the sentence-ender as the ultimate head in a sentence. In Korean, a sentence marker (postpositional marker for declarative, interrogative, imperative, honorific declarative, etc.) is usually obligatory in a sentence. These sentence-ending markers show that the whole construction is a sentence (1).

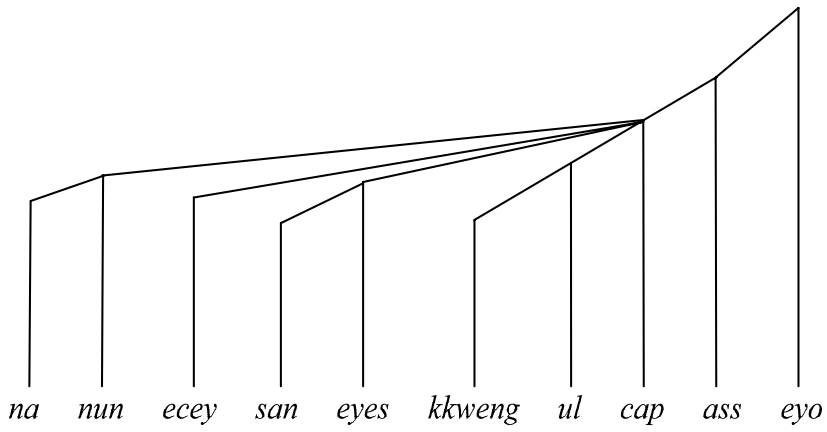
The sentence-ending marker optionally takes the tense marker. Therefore, according to the head and dependent criteria above (1), the tense marker is a dependent of the sentence-ending marker. The tense marker contains the inflection and it subcategorizes a verb. Therefore, the tense marker is the head of the verb stem (4).

The verb stem determines the valency structure and, in the case of the verb *cap*, it requires a subject, object, and optionally an adjunct. The subject and object are marked by the postpositional case marker and the adjunct can be marked by the adverbial marker. Thus, the verb stem is the head of the case marker and the adverbial marker (3).

The particles mentioned in 5.1 attach to either a verb or a noun. In the sentences above, the case marker and the adverbial marker are obligatory (1), contain inflections (4), and determine external relations in a sentence (3). Therefore, particles are heads of the noun or verb to which they attach.

The overall dependency tree of the sentence in (22) is shown in (24).

(24)



The head and dependent analysis above can lead to controversies. First, in casual speech, Korean speakers tend to omit the case marker and the hearer can still understand the sentence by context (Kang, Park, Yoon & Kwon 2002: 10). However, it is usually the case that it is either the nominative or accusative case marker that is omitted. Moreover, if the verb is intransitive,

then the parser can analyze the noun without any postpositional particle as the subject. If the verb is transitive, it is normally the case that the accusative case marker is omitted.

(25) a. ***chulsoo*** *jib eyse* ***ja*** *- n - da*
Chulsoo home at **sleep** pres dec
'Chulsoo sleeps at home.'

b. *chulsoo ga* ***sagwa*** ***mek*** *- nun - da*
Chulsoo nom **apple** **eat** pres dec
'Chulsoo eats an apple.'

c. ?***chulsoo*** *sagwa lul* ***mek*** *- nun -da*
Chulsoo apple acc **eat** pres dec
'Chulsoo eats an apple.'

d. ??***chulsoo*** ***sagwa*** ***mek*** *- nun -da*
Chulsoo **apple** **eat** pres dec
'Chulsoo eats an apple.'

Since the verb *ja* is intransitive in (25a), the nominative case marker can be omitted and still be understood as the subject. Similarly in (25b), the verb *mek* is transitive, thus the accusative case marker may be omitted and still understood as the object. However, by omitting the nominative case marker in (25c), the sentence sounds unnatural, and in (25d), the sentence becomes more unnatural if both case markers are omitted. Therefore, KorPar is implemented to parse a sentence that is composed of an intransitive verb and a noun without a case marker, as a subject. Furthermore, a sentence composed of a transitive verb and a noun without a case marker is parsed as an object.

The second controversy is that the noun and verb determine the morphosyntactic structure of the case marker and tense marker, respectively. As mentioned in 5.1, the ending features of the noun/case marker dependency pair and verb/tense marker dependency pair depend on the

noun or verb. In that case, based on the criterion (5), the noun and verb should be the head of the postpositional particles.

Therefore, we need to bear in mind that not all criteria can be met, and the main goal is to establish dependency rules that are efficient and consistent for implementation.

5.2.3. GULPFEATURES AGREEMENT

Once the head and dependent pairs are linked based on their **PartofSpeech**, the **GULPFeatures** are checked. For example, the ending features of a noun and the postpositional case marker need to match. As explained in 5.1, the value of the **ending** feature is decided by the dependent. For example, if the noun ends with a consonant, then the case marker starts with a vowel and that case marker has the **ending** feature value consonant.

(26) **% noun/postpositional case marker (pp)**
check_dh([N,_,_,n,X],[NN,_,_,pp,Y]) :- !, X = ending:End, Y = ending:End, NN is N +1.

The **GULPFeatures** agreement and order restrictions are stated in the **Condition** (see (21)) of the dependency rules.

5.2.4. SUBCATEGORIZATIONS

The **ListofDependents** atom in the **Node** list of each word plays an important role in checking subcategorization features. The parser keeps track of the "so far" dependency relation each time a word enters the parsing loop. The dependency rule checks the **ListofDependents** for the number, type, and order of subcategories.

For example, the subcategorizations of intransitive and transitive verbs are checked. As a result, sentences with two nominative case markers and only one verb are parsed as ungrammatical sentences. (Sentences with verbs that have the stative verb feature are exceptions: See (81).)

(27) **check_dh([N3,_,_,pp,X],[N2,[[N1,_,_,pp,Z]],_,v,Y]) :- !, Y = subcat:2,**
\+ (X = case:Case, Z = case:Case),
N2 > N1, N2 > N3.

The dependency rule for a verb that has the subcategorization feature value of 2 (transitive verb) and the case of both the subject and object being realized is shown in (27). Therefore, when **v** has **pp** (in **N1** position in a sentence) as a dependent and wants to check if that **v** can have **pp** (in **N3** position in a sentence) as another dependent, it will succeed if the values of **case** features are not the same between the two **pp'**s and if the order restrictions are met. The second atom in the **Node** list of the verb, **[[N1,_,_,pp,Z]]**, is the **ListofDependents** and is used to check subcategorizations.

5.2.5. EXAMPLE OF GRAMMAR RULES

The dependency grammar rules are categorized in 23 cases based on the categories of the **PartofSpeech** in the lexicon. I will provide examples and detail explanation of three cases: rules related to nouns, verbs, and **c** (either complementizers or conjunctions).

5.2.5.1. GRAMMAR RULES RELATED TO NOUNS

(28) **% noun/postpositional case marker (pp)**
check_dh([N,_,_,n,X],[NN,_,_,pp,Y]) :- !, X = ending:End, Y = ending:End,
NN is N +1.

The postpositional case marker is the head of the noun and the **ending** feature needs to be matched. There is an "adjacency restriction" for the noun and postpositional case marker pairs. (Reptition of (26).)

(29) **% n/n**
check_dh([N,_,_,n,X],[NN,_,_,n,Y]) :- !, NN is N +1.
% num(numeral)/n
check_dh([N,_,_,num,X],[NN,_,_,n,Y]) :- !, NN is N +1.
% adjective/noun
check_dh([N,_,_,adj,_],[NN,_,_,n,_]) :- !, NN is N+1.

There can be three different categories of **PartofSpeech** modifying a noun: noun (for consecutive nouns), numeral, and adjective. They all have the order restriction requiring them to be adjacent to the noun.

(30) ***hangul*** ***gongbu*** *nun* *jungyoha* *da*
Korean **study** nom important dec
'Studying Korean is important.'

There are many cases of compound nouns (consecutive nouns) in Korean, as in (30). To facilitate consistency in the order of the head and dependent in the grammar rules of KorPar, the previous noun is the dependent of the following noun.

(31) **% n/gen**
check_dh([N,_,_,n,X],[NN,_,_,gen,Y]) :- !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,_],[NN,_,_,gen,_]) :- !, NN is N +1.
%gen/n
check_dh([N,_,_,gen,X],[NN,_,_,n,Y]) :- !, NN > N.

In (31), the genitive case marker is the head of the preceding noun, matching the ending feature if necessary, and the following noun is the head of the genitive marker.

(32) ***changsoo*** ***-euy*** ***chingu*** *-dul* *-i* *o* *-n* *-da*
Changsoo **gen** **friend** pl nom come pres dec
'Changsoo's friends are coming.'

The genitive marker *euy* is the head of the preceding noun *changsoo* and in turn, it is the dependent of the following noun *chingu*.

(33) **% n/advp(adverbial post marker)**
check_dh([N,_,_,n,X],[NN,_,_,advp,Y]) :- !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,X],[NN,_,_,advp,Y]) :- !, NN is N +1.
% advp/v
check_dh([N,_,_,advp,X],[NN,_,_,v,Y]) :- !, NN > N.
% advp/p
check_dh([N,_,_,advp,X],[NN,_,_,p,Y]) :- !, NN is N +1.
%advp/pp
check_dh([N,_,_,advp,X],[NN,_,_,pp,Y]) :- !, NN is N +1.

The adverbial postpositional markers attach to a noun with the ending features matched and the "adjacency restriction" observed. The head of the adverbial postpositional marker can be either a verb (34a), a postpositional case marker (34c), or a sentence-ending marker (34b).

(34) a. *ge* *-ga* *hakkyo* ***-ro*** ***ga*** *-n* *-da*
he nom school **to(advp)** **go** pres dec
'He goes to school.'

b. *ge* *-ga* *senmul* *-ul* *bat* *-un* *-geut* *-un* *emma* ***-robute*** ***-da***
he nom present acc get vcn thing nom mom f**rom(advp)** **dec**
'The person he got the present from was mom.'

c. *hakkyo* ***-eyse*** ***-nun*** *joyoung* *-he-ya* *ha* *-n* *-da*
school **at(advp)** **nom** quiet must do pres dec
'You need to be quiet at school.'

(35) **% v/vnp(post marker that makes verb into noun)**

check_dh([N,_,_,v,X],[NN,_,_,vnp,Y]):- !, X = ending:End, Y = ending:End, NN is N+1.
check_dh([N,_,_,v,X],[NN,_,_,vnp,Y]):- !, NN is N+1.
% tp/vnp
check_dh([N,_,_,tp,X],[NN,_,_,vnp,Y]):- !, NN is N+1.
%vnp/pp
check_dh([N,_,_,vnp,X],[NN,_,_,pp,Y]):- !, X = ending:End , Y = ending:End, NN is N+1.
%n/ nnp
check_dh([N,_,_,n,_],[NN,_,_,nnp,_]):- !, NN is N+1.
% nnp/v(ha)
check_dh([N,_,_,nnp,_],[NN,_,ha,v,_]):- !, NN is N+1.
%nnp/cp(i)
check_dh([N,_,_,nnp,_],[NN,_,_,cp,_]):- !, NN is N+1.

In (35), the dependency rules for two kinds of postpositional nominalizers, one that attaches to a verb (**vnp**) and one that attaches to a noun (**nnp**), are listed. They both must be adjacent to their dependents; verb (36a) or tense marker (36b) for **vnp** and noun (36c) for **nnp**.

(36) a. ***galechi*** ***-m*** *-un* *jungyoha* *-da*
teach **vnp** nom important dec
'Teaching is important.'

b. *galechi* ***-ss*** ***-um*** *-un* *bunmyoungha* *-da*
teach **past** **vnp** nom sure dec
'It is sure that _ taught.'

c. *saero* *-un* *beb* *-ul* ***hapbeb*** ***-wha*** *ha* *-da*
new vcn law acc **legal** **nnp** do dec
'Making the new law legal.'

5.2.5.2. GRAMMAR RULES RELATED TO VERBS

(37) **% verb/p**
check_dh([N,_,_,v,_],[NN,_,_,p,_]) :- !, NN is N+1.

% tp/p
check_dh([N,_,_,tp,_],[NN,_,_,p,_]) :- !, NN is N +1.
% n/p noun ending in vow only
check_dh([N,_,_,n,X],[NN,_,_,p,_]) :- !, X = ending:vow, NN is N + 1.
% p(sentence-endingmarker)/postpositional marker(pp)
check_dh([N,_,_,p,X],[NN,_,_,pp,Y]) :- !, X = ending:End, Y = ending:End, NN is N +1.

As mentioned in 5.2.2, the ultimate head (a word that does not have a head) of a sentence in Korean is the sentence-ending marker. The sentence-ending marker is preceded with either a tense marker **tp** or the infinitive verb form **v.**

There are two exceptional cases. First, if a noun ends with a vowel, the copular verb *-i* can be omitted. Therefore, the sentence-ending marker can be preceded by a noun ending with a vowel, as in (38a). Second, if there is a subordinate clause, the sentence-ending marker of the subordinate clause is the dependent of the following postpositional case marker, as in (38b).

(38) a. *ge -ga ga -n got -un* ***bada -da***
he nom go vcn place nom **sea dec**
'The place that he went is the sea.'

b. *ge -ga ha -n -mal -un sarangha -n* ***-da -ga*** *ani -da*
he nom say vcn word nom to love pres **dec nom** neg dec
'He did not say that he loved me.'

(39) **% verb/tp**
check_dh([N,_,_,v,X],[NN,_,_,tp,Y]) :- !, X = ending:End, Y = ending:End, NN is N + 1.
check_dh([N,_,_,v,_],[NN,_,_,tp,_]) :- !, NN is N + 1.

The verb is a dependent of the following tense marker, with the ending features matched and the order restrictions applied, as listed in (39).

(40) **% v/serv & serv/verb**
check_dh([N,_,_,v,_],[NN,_,_,serv,_]) :- !, NN is N + 1.
check_dh([N,_,_,serv,_],[NN,_,_,v,_]) :- !, NN is N + 1.

The serial verbs are connected to the postpositional particle, *-e/-go,* and are the head of the previous verb and the dependent of the following verb. (See also 5.4.5.)

(41) **% n/cp(copular postmarkers)**
check_dh([N,_,_,n,_],[NN,_,_,cp,_]) :- !, NN is N +1.
% cp/tp
check_dh([N,_,_,cp,_],[NN,_,_,tp,_]) :- !, NN is N +1.
%cp/p
check_dh([N,_,_,cp,_],[NN,_,_,p,_]) :- !, NN is N +1.
% cp/vcn
check_dh([N,_,_,cp,_],[NN,_,_,vcn,_]) :- !, NN is N +1.

The copular verb **cp** has a similar meaning to the copular verb *be* in English. It is a head of a noun. The heads of copular verbs are similar to that of a verb: tense marker, sentence-ending marker, or the complementizer **vcn,** as in (42).

(42) *ge -nun* ***haksaeng*** ***-i*** ***-n*** *geut -euro al -da*
he nom **student** **cp** **vcn** thing as know dec
'I know that he is a student.'

The other kind of copular verb **ccp** has a similar meaning to the verb *become* in English. There are two different dependency relations that categorize this copular verb that are separate

from verbs with the **subcat** feature value of 1 and the **feat** value dynamic or verbs with the **subcat** feature value of 2.

First, similar to **cp**, **ccp** can be preceded by a noun.

(43) **% n/ccp**
check_dh([N,_,_,n,_],[NN,_,_,ccp,_]) :- !, NN is N +1.

Second, similar to **v** with **feat** value sta, **ccp** can have two nominative case markers as dependents.

(44) **% two pp/ccp**
check_dh([N,_,_,pp,X],[NN,[[L,_,_,pp,Z]],_,M, Y]) :- !,
((M = ccp) ; (M = v , Y = feat:sta)),
\+ (X = case:acc, Z = case:acc),
N < NN, L < NN.
% ccp/tp
check_dh([N,_,_,M,Y],[NN,_,_,tp,_]) :- !, ((M = ccp) ; (M = v , Y = feat:sta)),
NN is N +1.
%ccp/p
check_dh([N,_,_,M,Y],[NN,_,_,p,_]) :- !, ((M = ccp) ; (M = v , Y = feat:sta)),
NN is N +1.

The condition, **\+ (X = case:acc, Z = case:acc)** means that the two postpositional case markers, which are dependents of **ccp**, cannot both be accusative case markers. In other words, they can both be nominative case markers.

(45) a. *ge* ***-ga*** *whalgi-cha* *-n* *saram* ***-i*** ***dwe*** *-ss* *-da*
he **nom** active vcn person **nom** **ccp** past dec
'He became an active person.'

b. *ge* ***-nun*** *ki* ***-ga*** ***ke*** *-da*
he **nom** height **nom** **tall(stative v)** dec
'He is tall.'

As listed in Section 5.2.2, the verb can be the head of one or two **pp**(s) and/or the "optional dependents," **scs, ncv, vcv, serv, pcv, advp, adv, nnp**. Therefore, according to the difference in number and type of subcategorization, verbs are categorized as having either the **subcat** feature value of 1 or 2. A verb with a value of 1 can have only one nominative case marker and/or "optional dependent." In comparison, a verb with a value of 2 can have either a nominative and/or accusative case marker as a dependent(s), as well as "optional dependents."

A verb with the **subcat** feature value of 1 is categorized in two different cases: **feat** value sta (stative) or dyn (dynamic). As mentioned above, a verb with the value sta can have two nominative case markers as dependents, similar to **ccp**. (See also 5.4.6.2.)

Several dependency rules for verbs that are heads of one postpositional case marker and/or other "optional dependents" are listed in (46). The verb with the **subcat** feature value of 1 is linked with a nominative case marker and the verb with the value of 2 is linked with either a nominative or accusative case marker. The order of the postpositional case marker and other "optional dependent(s)" are free, thus the parser will try all different combinations of orders later in the algorithm.

(46) **%case marker/verb subcat:1 OR subcat:2 that has one dep. realized.**

```
        check_dh([N,_,_,pp,X],[NN,[],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                        (X = case:nom ;X = case: nom-topicalization));
                                        (Y = subcat:2, X = case:acc)), N < NN.
% verb subcat:1  or 2 / pp& adv or advp
        check_dh([N,_,_,pp,X],[NN,[[M,_,_,adv,_]],_,v,Y]) :- !,
                                        (((Y = subcat:1 ; Y = subcat:2),
                                          (X = case:nom ;
                                           X = case: nom-topicalization;
                                           X = case:nom-also));
                                         (Y = subcat:2, X = case:acc)),
                                          N < NN, M < NN.
% verb subcat:1 or 2 / pp& adv or advp
        check_dh([N,_,_,pp,X],[NN,[[M,_,_,advp,_]],_,v,Y]) :- !,
                                        (((Y = subcat:1 ; Y = subcat:2),
```

```
                                        (X = case:nom ;
                                         X = case: nom-topicalization;
                                         X = case:nom-also));
                                        (Y = subcat:2, X = case:acc)),
                                         N < NN, M < NN.
% verb subcat:1  or 2/ pp & ncv
      check_dh([N,_,_,pp,X],[NN,[[M,_,_,ncv,_]],_,v,Y]) :- !,
                                    (((Y = subcat:1 ; Y = subcat:2),
                                      (X = case:nom ;
                                       X = case: nom-topicalization;
                                       X = case:nom-also));
                                      (Y = subcat:2, X = case:acc)),
                                       N < NN, M < NN.
% verb subcat:1 or 2 / pp & advp & ncv
      check_dh([N,_,_,pp,X],[NN,[[M,_,_,advp,_],[O,_,_,ncv,_]],_,v,Y]) :- !,
                                    (((Y = subcat:1 ; Y = subcat:2),
                                      (X = case:nom ;
                                        X = case: nom-topicalization));
                                      (Y = subcat:2, X = case:acc)),
                                       N < NN, M < NN, O < NN.
```

(47) a. *gwaja* ***-lul*** *gagyae* ***-eyse*** ***sa*** *-ss* *-da*
cookie **acc** **s**tore **at(advp)** **buy** past dec
'He bought the cookie at the store.'

b. *ge* ***-nun*** *san* ***-edo*** ***ga*** *-n* *-da*
he **nom** mountain **also(ncv)** **go** pres dec
'He also goes to the mountain.'

In (47a), the verb *sa* has the **subcat** value of 2 and is a head of the accusative case marker *lul* and **advp** *eyse*. In (44b), the verb *ga* has the **subcat** value of 1 and is a head of the nominative case marker *nun* and **ncv** *edo*.

A verb with the subcategorization feature value of 2 can have both the nominative and accusative postpositional case marker realized with or without other "optional dependents." The two postpositional case markers cannot have the same value of cases, as is stated in the condition of the example rules below.

(48) **\+ (X = case:Case,Z = case:Case)**

The following are examples of rules for verbs with the **subcat** feature value of 2.

(49) **%case marker/verb subcat:2(acc & nom)**
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Z]],_,v,Y]) :- !, (Y = subcat:2),
\+ ((X = case:nom,Z = case:nom-topicalization);
(X = case:nom-topicalization, Z = case:nom)), N < NN, M < NN.
%case marker/verb subcat:2 that has both subject, object, and one of the optional dependents.
check_dh([N,_,_,pp,X],[NN,[[M,_,_,ZZ,_],[L,_,_,pp,Z]],_,v,Y]) :- !, Y = subcat:2,
(ZZ = adv; ZZ = advp; ZZ = ncv),
\+ (X = case:Case,Z = case:Case),
N < NN, M < NN, L < NN.
%verb subcat:2 that has subject,object,adv,advp
check_dh([N,_,_,pp,X],[NN,[[L,_,_,adv,_],[O,_,_,advp,_],[M,_,_,pp,Z]],_,v,Y]) :- !,
Y = subcat:2,
\+ (X = case:Case,Z = case:Case),
N < NN, M < NN, L < NN, O < NN.

(50) *ge* ***-nun*** *sagwa* ***-lul*** ***manni*** ***mek*** *-ss* *-da*
he **nom** apple **acc** **a lot (adv)** **eat** past dec
'He ate a lot of apples.'

The verb *mek* has the **subcat** value 2, and is the head of the nominative case marker *nun*, accusative case marker *lul*, and the adverb *manni*.

5.2.5.3. GRAMMAR RULES RELATED TO C (COMPLEMENTIZER/CONJUNCTION)

As mentioned in 5.1, the postpositional **c** is categorized into five subcategories based on their dependency relations. These postpostional particles either introduce a subordinate clause or connect one word to another.

(51) **%noun/ncv**
check_dh([N,_,_,n,X],[NN,_,_,ncv,Y]) :- !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,_],[NN,_,_,ncv,_]) :- !, NN is N +1.
% ncv(complementizer)/v
check_dh([N,_,_,ncv,X],[NN,_,_,v,Y]) :- !, NN > N.

In (52), the noun *ganan* is a dependent of **ncv** and the **v** *senggongha* is the head. The noun and the **ncv** need to agree in ending features, but not in all cases (depending on the lexical item of **ncv**).

(52) ***ganan*** ***-edo-bulguhago*** *ge* *-nun* ***senggongha*** *-ss* *-da*
poverty **in spite of (ncv)** he nom **succeed** past dec
'In spite of his poverty, he succeeded.'

(53) **% noun/scs**
check_dh([N,_,_,n,X],[NN,_,_,scs,Y]) :- !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,_],[NN,_,_,scs,_]) :- !, NN is N +1.
%scs/noun
check_dh([N,[[M,_,_,n,_]],_,scs,_],[NN,_,_,n,_]) :- !, N is M + 1, N < NN.
%verb/scs
check_dh([N,_,_,v,_],[NN,_,_,scs,_]) :- !, NN is N +1.
%scs/verb
check_dh([N,[[M,_,_,v,_]],_,scs,_],[NN,_,_,v,_]) :- !, N is M + 1, N < NN.

The rules related to **scs** is listed in (53). The dependent and the head have the same lexical category for **scs: n, v, cp,** or **vnp**. The rules when the **scs** is a dependent state this condition. The dependent of the **scs** has the same lexical category as the head of the **scs**.

(54) a. ***gwail*** ***-gwa*** ***chaeso*** *-nun* *mom* *-ey* *jot* *-da*
fruit **and(scs)** **vegetable** nom body to good dec
'Fruits and vegetables are good for your health.'

b. *ge -ga* ***gochi -go*** ***ga*** *-ss -da*
he nom **fix** **and(scs)** **go** past dec
'He fixed it and then left.'

The head and dependent of **scs** in (54a) are the nouns *gwail* and *chaeso*, whereas in (54b), they are the verbs *gochi* and *ga.*

The dependency rules related to **vcv** is shown in (55).

(55) **%v/vcv**

```
check_dh([N,_,_,v,X],[NN,_,_,vcv,Y]) :-  !, X = ending:End, Y = ending:End,
                                          NN is N +1.
check_dh([N,_,_,v,_],[NN,_,_,vcv,_]) :-  !, NN is N +1.

% cp/vcv
check_dh([N,_,_,cp,X],[NN,_,_,vcv,Y]) :-  !, NN is N +1.
% ccp/vcv
check_dh([N,_,_,ccp,X],[NN,_,_,vcv,Y]) :-  !, NN is N +1.
% vcv/v
check_dh([N,_,_,vcv,X],[NN,_,_,v,Y]) :-  !, NN > N.
% vcv/cp
check_dh([N,_,_,vcv,X],[NN,_,_,cp,Y]) :-  !, NN > N.
% vcv/ccp
check_dh([N,_,_,vcv,X],[NN,_,_,ccp,Y]) :-  !, NN > N.
```

Since the two kinds of copular verb *-i* (**cp**), and *-dwe* (**ccp**) are also verbs, they can be the head and/or the dependent of **vcv**. The difference with **scs** is that a single **vcv** does not need to have the same lexical category for its dependent and head as (56).

(56) *eli -saedae* ***-i - l- surok*** *young-e -lul jal* ***ha*** *-n -da*
young generation **be(cp) the more(vcv)** English acc well **do(v)** pres dec
'The younger the generation is, the better they speak English.'

The **vcv** complementizer *l-surok* is the head of the copular verb *-i* and the main verb *ha* is the head of the **vcv**.

(57) **%verb/vcn**
check_dh([N,_,_,v,X],[NN,_,_,vcn,Y]):- !, X = ending:End, Y = ending1:End, NN is N +1.
%p/vcn
check_dh([N,_,_,p,X],[NN,_,_,vcn,Y]):- !, NN is N +1.
%vcn/n
check_dh([N,_,_,vcn,_],[NN,_,_,n,_]):- !, NN > N.

The complementizer **vcn** is similar to "that" in English and is used frequently to modify nouns. Therefore, there are controversies on the analysis of adjectives and relative clauses introduced by **vcn** in Korean (See also 5.4.6). The verb or the sentence-ending marker can be the dependent of **vcn** and the modified noun is the head.

(58) a.

ge -nun	***eli***	***-n***	***saram***	*-i*	*-da*
he nom	**young (verb)**	**vcn**	**person**	be(cp)	dec

'He is a young person.'

b.

ge -nun	***eli***	***-da***
he nom	**young (verb)**	**dec**

'He is young.'

A noun's modifier can modify either attributively (58a) or predicatively (58b). KorPar categorizes all modifiers (normally classified as "adjective") that can modify a noun predicatively as verbs with the **subcat** feature value of 1 and the **feat** feature value sta. Therefore, *eli-* is a verb with the **subcat** feature value of 1 and the **feat** feature value sta. The **vcn** attaches to the verb *eli-* and modifies the noun *saram*. In addition, as mentioned above, the verb *eli-* can have two nominative case markers as in (59).

(59) *ge* ***-nun*** *na-i* ***-ga*** ***eli*** *-da*
he **nom** age **nom** **young** dec
'He is young (in age).'

The verb *eli* is the head of two nominative markers: *nun* and *ga*.

(60) **%pcv(complementizer)/v**
check_dh([N,_,_,pcv,X],[NN,_,_,v,Y]) :- !, NN is N +1.
% p/pcv(complementizer)
check_dh([N,_,_,p,X],[NN,_,_,pcv,Y]) :- !, NN is N +1.

The last subcategory of **c** is **pcv** and has the similar meaning and function as "that, although" in English. The sentence-ending marker of the subordinate clause (*da*) is the dependent of **pcv** (*go*) and the main verb (*malha*) is the head of it in (61).

(61) *ge* *-ga* *ga* *-n* ***-da*** ***-go*** ***malha*** *-n* *-da*
he nom go pres **dec** **that(pcv)** **say** pres dec
'_ says that he is going.'

5.3. ALGORITHM

The overall algorithm of the parser is shown in Figure 1 in pseudo-code (compare with Covington 2001 and Covington 2003).

Data Structures

InputList = list of words in a sentence to be parsed

HeadList = empty list to contain nodes (subtrees)

Algorithm

Note 1: Algorithm is implemented nondeterministically in Prolog and can backtrack to any alternative. Lexical entries, dependency rules, and the algorithm introduce nondeterminism.

For each word *w1* in InputList:
    Look up *w1* in lexicon
    Create a **Node** for w1
    For each node *w2* in HeadList:
        If *w2* can be a dependent of *w1*:
            Remove *w2* from HeadList
            Attach *w2* as a dependent of *w1*
        End if
    End for
    Add *w1* **Node** to the beginning of HeadList
End for

Note 2: HeadList now contains one node, which includes the complete dependency tree. Because Korean is head-final, the items in the HeadList can only be dependents of w1, not vice versa.

Figure 1. Overall Algorithm of KorPar (in pseudo-code)

The input of KorPar is a list of words composing the sentence that needs to be parsed. The output would be a notation of each word, showing the dependency relation by indentation and displaying the features in GULP syntax. The actual input and output of the sentence (15a) in 4.3 are shown in (62).

(62) **?- try([gang-a-ji,ga,go-yang-i,lul,chot,nun,da]).**

```
7 [da, p, sem:declarative]
    6   [nun, tp, ending:con..tense:pres]
          5     [chot, v, ending:con..sem:chase..subcat:2..feat:dyn]
                4       [lul, pp, ending:vow..case:acc..num:sing]
                  3           [go-yang-i, n, ending:vow..sem:cat]
                2       [ga, pp, ending:vow..case:nom..num:sing]
                  1           [gang-a-ji, n, ending:vow..sem:dog]
```

The output of the parsing algorithm itself is a Prolog term. The features must be retranslated from a Prolog term to a readable structure. The built-in predicate of the GULP system, **display_feature_structure**, displays the feature structure in a readable tabular format. Therefore, the number in a sentence is indicated for each word and the phonetic sound, part of speech, and different **GULPFeatures** (**ending**, **tense**, **case**, **sem**, etc) are displayed.

Figure 2 shows the parsing process of the same sentence (15a) by listing the changes of the InputList and HeadList.

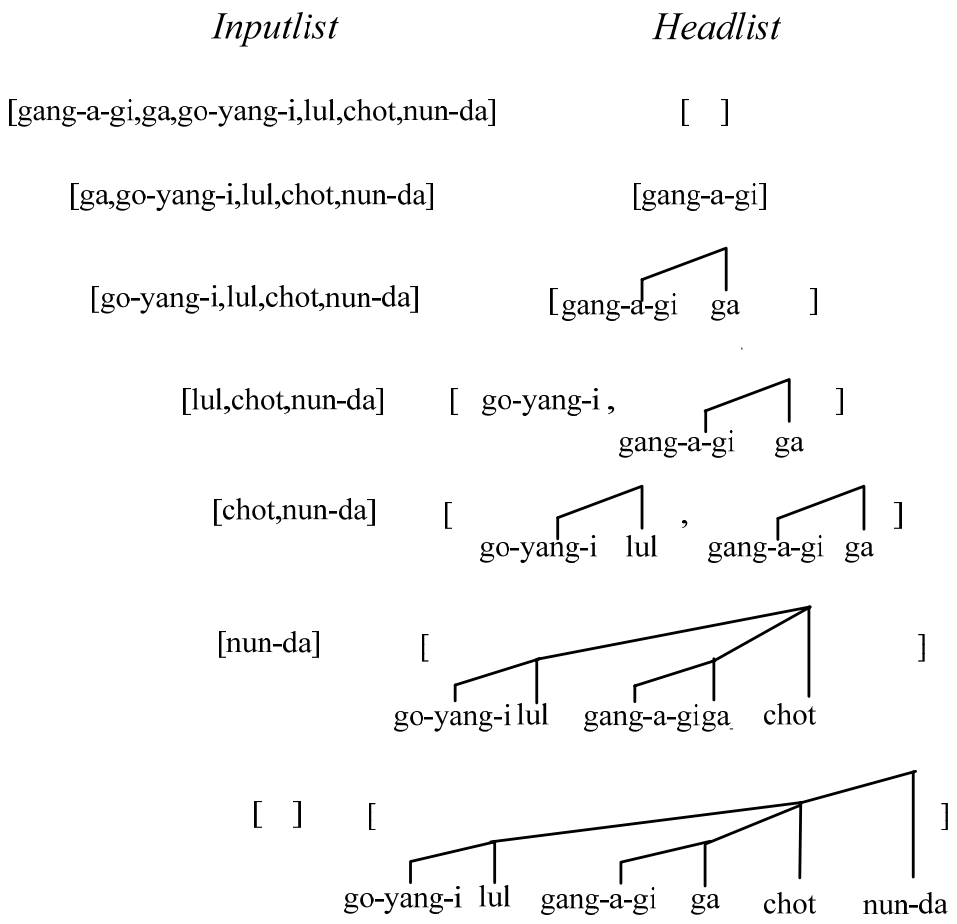


Figure 2. Parsing Process of Sentence (12a)

The parser starts with an empty HeadList and ends with a HeadList that contains the complete dependency tree in the **ListofDependents** atom. The parsing algorithm ends when the InpuList is empty.

The algorithm itself has roughly three major predicates. The main predicate, **parse,** initializes the parsing algorithm. The second predicate, **parse_loop,** loops over all the steps. The third predicate, **parse_node,** links the dependency pairs.

5.3.1. parse(+InputList, -Result)

(63) **parse(InputList,Result) :- parse_loop(1,InputList,[],Result).**

The parsing algorithm begins with the predicate **parse**. This predicate accepts the input list of words and returns the result passed back from the predicate **parse_loop**. The looping predicate is initialized with a **Number** of 1 and an empty HeadList (a list of words that seem to be heads).

5.3.2. parse_loop(+NumberinSentence,+InputList,+HeadList,-Result).

The predicate **parse_loop** repeats the looping steps shown in Figure 1 until the InputList is empty. The loop will stop when only one node in the HeadList remains; this is the ultimate head. The second argument in this **Node**, the **ListofDependents** atom, contains the overall dependency tree.

(64) **parse_loop(Number,[Word|InputList],HeadList,Result) :-**
look_for_word(Number,Word,Node), %look up the lexicon and
change to a list, Node
parse_node(Node,HeadList,NewHeadList), %try to find a dependency
NewNumber is Number + 1, %go to the next word
parse_loop(NewNumber,InputList,NewHeadList,Result).

parse_loop(_,[],[H],[H]). % no more words to parse; so Result = HeadList.

As illustrated in Figure 1, KorPar provides a consistent analysis of the order of head and dependent pairs. Since Korean is a head-final language, the head always follows the dependent. Specifically, each word in the HeadList can only be a dependent of the following input word and the dependency pair is linked by the predicate **parse_node**. Chung (2004) and Kim, Byeon, and Oh (1999) set the same dependency constraint in order to avoid totally free word order that results in incorrect analysis.

5.3.3. parse_node(+Node,+HeadList,-NewHeadList).

The predicate **parse_node** links the head and dependent pairs based on the dependency rules. Each time the dependency relation is checked, the HeadList is modified. There are three cases of "HeadList modification."

First, if the HeadList is empty, then current input word is simply added to the beginning of the HeadList. This case applies to the first input word and the end of the recursive steps of the predicate **parse_node**.

(65) **parse_node(Node,[ ],[Node]).**

The next word in the InputList is then sent to the **parse_loop** predicate.

The second case of HeadList modification arises because the dependency rules in the grammar state only one kind of order combination for the words in the **ListofDependents**. However, since Korean is a partially free-word-order language, almost all dependents of the verb and the attached particles are free in order. Therefore, when checking dependency relations, different combinations of the order of dependents must be considered.

(66)
```
%if the current Node does not have any "so far" dependents.
        parse_node(Node,[Head|HeadList],NewHeadList) :-
                Node = [N,[],W|X],
                Head = [NN,_,WW|Y],
                check_dh(Head,Node),     % Head is the dependent here
                NewNode = [N,[Head],W|X], % Head added to the ListofDependents
                parse_node(NewNode,HeadList,NewHeadList).%repeat until HeadList is empty
%if the current Node has 1 "so far" dependent.
        parse_node(Node,[Head|HeadList],NewHeadList) :-
                Node = [N,[D1],W|X],
                Head = [NN,_,WW|Y],
                check_dh(Head,Node),
                append([D1],[Head],NewD),
                NewNode = [N,NewD,W|X],
                parse_node(NewNode,HeadList,NewHeadList).
%if the current Node has 2 "so far" dependents.
        parse_node(Node,[Head|HeadList],NewHeadList) :-
                Node = [N,[D1,D2],W|X],
                Head = [NN,_,WW|Y],
                (check_dh(Head,Node);
                 check_dh(Head,[N,[D2,D1],W|X])  %try alternative order combination
                ),
                append([D1,D2],[Head],NewD),
                NewNode = [N,NewD,W|X],
                parse_node(NewNode,HeadList,NewHeadList).
% if the current Node has 3 "so far" dependents.
                parse_node(Node,[Head|HeadList],NewHeadList) :-
                Node = [N,[D1,D2,D3],W|X],
                Head = [NN,_,WW|Y],
                (check_dh(Head,Node);        %try alternative order combination
                check_dh(Head,[N,[D2,D1,D3],W|X]);
                check_dh(Head,[N,[D2,D3,D1],W|X]);
                check_dh(Head,[N,[D1,D3,D2],W|X]);
                check_dh(Head,[N,[D3,D1,D2],W|X]);
                check_dh(Head,[N,[D3,D2,D1],W|X])
                ),
                append([D1,D2,D3],[Head],NewD),
                NewNode = [N,NewD,W|X],
                parse_node(NewNode,HeadList,NewHeadList).
```

The cases where a single head has, at most, four dependents are listed above. Among the 100 test sentences parsed by KorPar, there were no more than six dependents for a single head. If there are more than seven dependents for a single head in a sentence, parsing becomes too complex and ambiguous, which is rare in natural language. Nevertheless, KorPar can easily be

modified to parse long, complex sentences with more than seven dependents for a single head by adding additional combinations for the predicate **parse_node**.

Finally, if the previous conditions were not met, it means that there is no head and dependent pair for the current **Node,** thus it is added to the beginning of the HeadList.

(67) **parse_node(Node,HeadList,[Node|HeadList]).**

By adding the current **Node** to the HeadList and preventing the parser from trying alternative cases, the projectivity constraint (no crossing of arcs) is obeyed. (See also 5.4.2.)

KorPar can also be modified to parse languages with non-projectivity (crossing of arcs) by substituting (67) with (68).

(68) **parse_node(Node,[Head|HeadList],[Head|NewHeadList]) :-**
parse_node(Node,HeadList,NewHeadList).

The modification in (68) allows the current node to look for its dependency pairs by trying alternative cases and eventually allowing the crossing of arcs.

5.4. TROUBLESHOOTING CASES

There were several problems that KorPar needed to troubleshoot in order to parse Korean. This chapter examines these problems and explains how KorPar solves them.

5.4.1. SUBCATEGORIZATION

Korean exhibits different characteristics in subcategorization from that of English. For example, if a verb subcategorizes two words (subject and object), this means that in Korean, the

verb can have at most two arguments. The subject and/or object can be omitted. Moreover, in addition to the numbers of subcategories being free, the orders of the subcategories are partially free.

Consequently, KorPar needs to monitor the number, type, and order of the subcategories. Contrary to the parsing algorithm introduced by Covington (2003), the word changes to a list **Node** with the second atom, **ListofDependents,** instantiated as an empty list, not as an anonymous variable.

(69) **look_for_word(Count,Word,Node) :- word(Word,WordFeatures),**
WordFeatures =.. X, Node = [Count,[ ],Word|X].

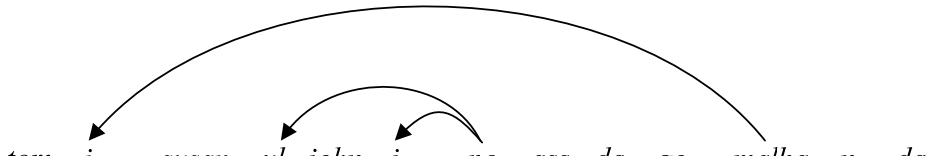
The **ListofDependents** (the second atom of **Node)** is instantiated with an empty list at the beginning of the parsing loop. Each time an input word links to its dependency pair by the predicate **parse_node**, the **ListofDependents** is updated by adding the dependent to the list. Then, when the next input word is checked for dependency linking, KorPar looks at the **ListofDependents** for subcategorization features.

For example, if a verb has the **subcat** feature value of 2, and has an accusative marker in the **ListofDependents,** KorPar will notice that another accusative case marker is not allowed by looking at the type, number, and order of the "so far" dependents of the verb in the **ListofDependents** atom.

5.4.2. LONG-DISTANCE DEPENDENCY

Because Korean is a partially free-word-order language, it has many cases of long-distance dependency. However, in most instances, the dependency links do not cross over each other, as in (70).

(70)

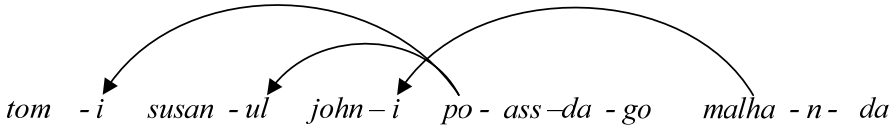


'Tom says that John saw Susan.'

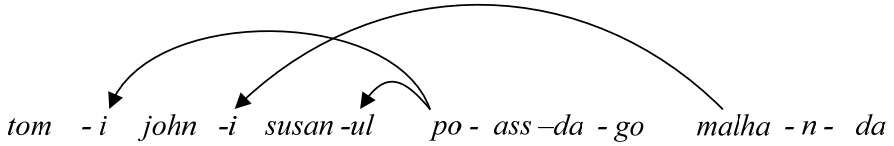
(example from Kwon and Yoon (1991))

In (70), there is an ambiguity in the choice of the nominative case marker for the two verbs *po* and *malha*. Since Korean tends to maintain the "no crossing of arcs" rule, the nominative case marker attached to *john* links with *po* and the nominative case marker attached to *tom* links with *malha*.

(71) a.



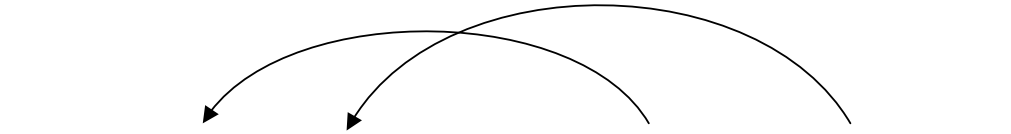
b.



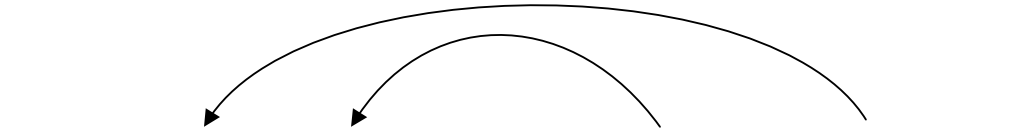
As in (71a), if the nominative case marker attached to *tom* links with the verb *po* and the one attached to *john* links with the verb *malha*, there will be a "cross of arcs," and these are not the correct dependency pairs. However, in (71b), although there is a "cross of arcs," the dependency pairs are linked correctly. According to Kwon and Yoon (1991), (71b) is a grammatical sentence. However, it takes longer to process the meaning of the sentence and it seems unnatural. In comparison, (70) is the preferred word order and dependency structure when conveying the same meaning.

Lee (2002) stated that Korean does not allow crossing of dependency links because it could convey the wrong meaning.

(72) a.



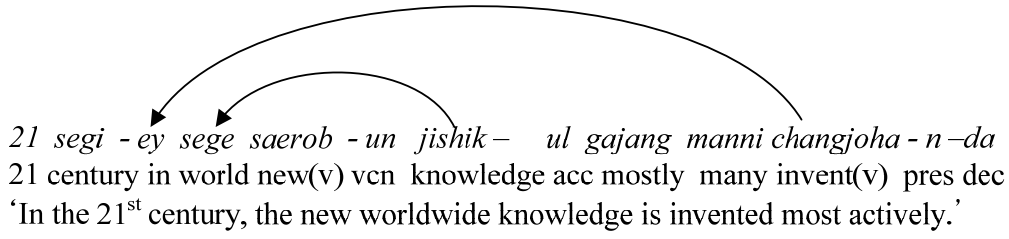
b.



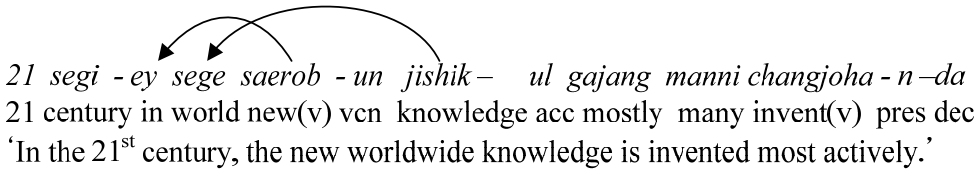
The dependency arcs in (72a) cross and the dependency pairs are linked incorrectly. Instead, the nominative topicalization marker *nun* should link with the verb *ha* and the nominative case marker *i* should link with the verb *malha*, resulting in no crossing of arcs, as in (72b).

Similarly in (73a), the **advp** *ey* links with the verb *changjoha,* not the verb *saerob*, since the noun *sege* links with the noun *jishik*. If *ey* links with *saerob*, then there would be two crossing arcs, as in (73b) and these are not the correct dependency pairs.

(73) a.



b.



KorPar manages to correctly link the head and dependent pairs without having any crossed arcs. In 5.3.3, the predicate **parse_node** has three cases of "HeadList modification". In the third case, where there are no head and dependent pairs to link, the current input word is added to the *beginning* of the HeadList. Adding it to the beginning of the HeadList allows the following input word to search for its dependent starting from the closest word.

For example, in (70), the closest nominative case marker of the verb *po* is the one that is attached to *john,* and they will therefore link together as a pair. In turn, the verb *malha* will be linked to the remaining nominative case marker, which is the one attached to *tom*. As a result, the arcs do not cross over one another.

Also, in (73a), the grammar rules do not link the verb *saerob* with the noun *sege*, and so the HeadList will have *sege* as the first item and the **advp** *ey* as the second item. By adding the input word to the beginning of the HeadList, the verb *saerob* and the **advp** *ey* are blocked from being linked as a pair and *ey* can later be linked with the verb *changjoha*. Figure 3 shows the parsing process of (73a).

Therefore, the right-to-left search process for head and dependent pairs not only prevents incorrect long-distance dependency, but also allows KorPar to be more efficient in time and space complexities.

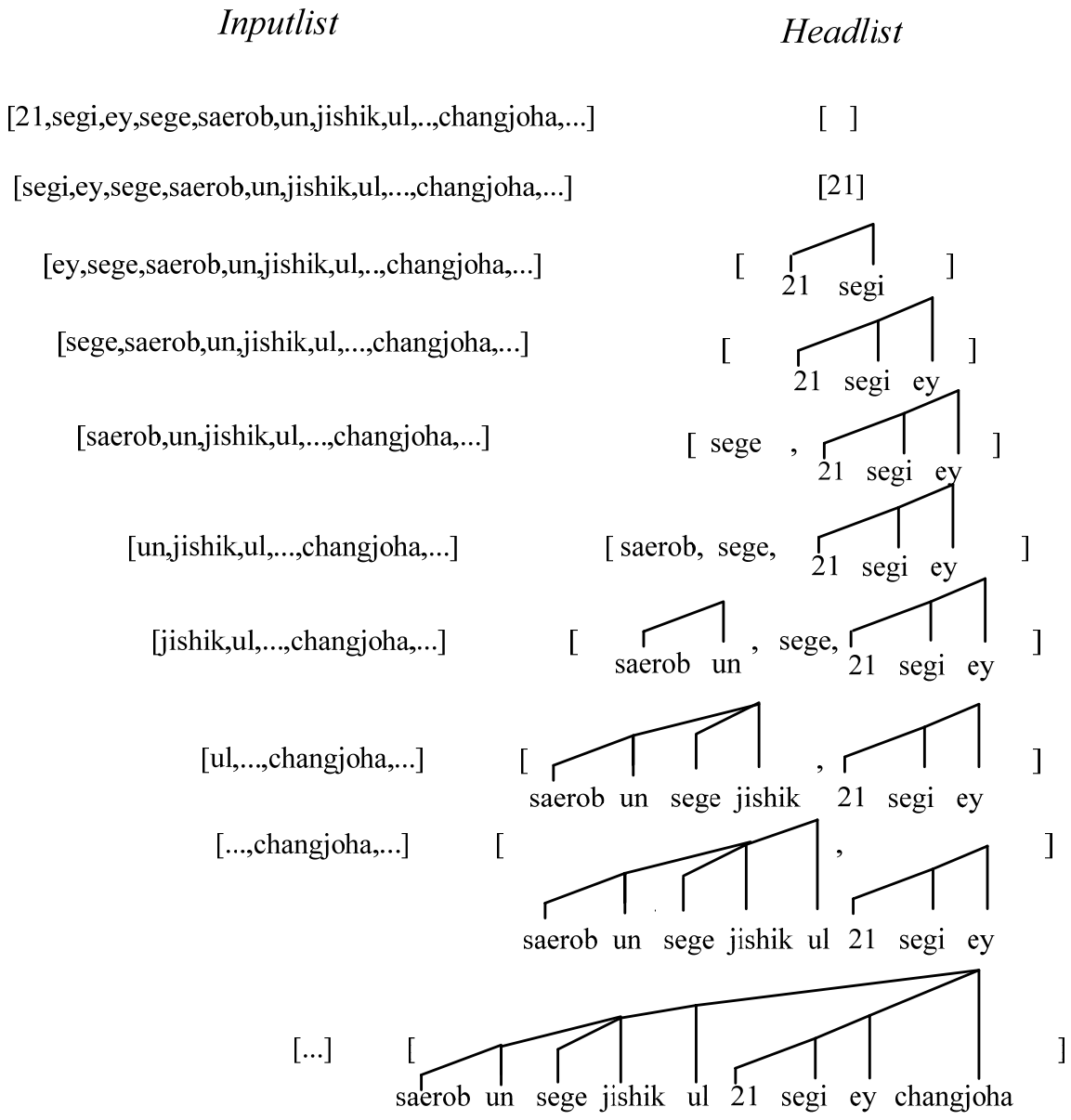


Figure 3. Parsing Process of Sentence (73a)

5.4.3. COMMON NOUNS, PROPER NOUNS AND PRONOUNS

Common nouns, proper nouns, and pronouns are categorized as nouns, since their syntactic relation and function are similar to nouns. In Korean, there can also be consecutive nouns without any "connector" to create compound words.

(74) ***seoul-dae enoh-hak ggwa I-hoyoung gyosoo eungu tim*** *-i balpyoha-ess -da*
seoul-univ. linguistics dept. I-hoyoung prof. research team nom announce past dec
'The research team of I-hoyoung professor in the linguistics department of Seoul University announced _.'

KorPar parses the sentence in (74) as (75). The overall dependency structure is shown by indentation and the part of speech and meaning are displayed for each word.

(75) **?- try([seoul-dae, enoh-hak,ggwa,i-hoyoung,gyosoo,eungu,tim,i,balpyoha,ss,da]).**

11 [da, p, sem:declarative]
10 [ss, tp, ending:vow..tense:past]
9 [balpyoha, v, ending:vow..sem:announce..subcat:2..feat:dyn]
8 [i, pp, ending:con..case:nom..num:sing]
7 [tim, n, ending:con..sem:team]
6 [eungu, n, ending:vow..sem:research]
5 [gyosoo, n, ending:vow..sem:professor]
4 [i-hoyoung, n, ending:con..sem:lee-hoyoung]
3 [ggwa, n, ending:vow..sem:department]
2 [enoh-hak, n, ending:con..sem:linguistics]
1 [seoul-dae, n, ending:vow..sem:seoul-university]

5.4.4. SERIAL VERB CONSTRUCTIONS

According to Stewart (2001), there are three descriptive definitions of serial verb construction (SVC): 1) two or more verbs and their arguments co-occur without any conjunction; 2) these verbs must share the same subject and (sometimes) the same object; and 3) there is usually a single tense or aspect specification for the verb.

Choi (2003) and Kang (1997) argued that the SVC is shown in Korean and the first verb in this construction changes its form by attaching a postpositional particle such as *-e* or *-go*.

(76) *ge* *-ga* *sagwa* *-lul* ***kkak*** ***-e*** ***-mek*** *-nun* *-da*
he nom apple acc **peel** **serv** **eat** pres dec
'He is peeling and then eating an apple.'

In (76), two verbs, *kkak* and *mek,* occur without any connector and the first verb changes its form by attaching *-e*. The two verbs share the subject *ge* and they are in the present tense.

Therefore, the lexicon of KorPar has a part of speech **serv** for SVCs in Korean. The dependent of **serv** is the preceding verb and the head is the following verb.

5.4.5. ADJECTIVES AND STATIVE VERBS IN KOREAN

Adjectives are parts of speech that modify a noun. It is believed that this part of speech is not universal (Beck 1999). Many researchers posit that Korean does not have a distinct and open class of adjectives (Choi 1971; Sohn 1999).

There are several morphosyntactic behaviors of the words that seem to modify nouns. For example, they can modify nouns attributively. However, they require a postpositional marker **vcn** (*-n*, *-nun*, *-l*, etc.) that has the structure of a relative clause (Kang, Park, Yoon & Kwon 2002: 10).

(77) a. *eli* *-n* *saram*
young vcn person
'A young person'

b. *ttena* *-n* *saram*
leave vcn person
'A person who left.'

The verbs *eli* and *ttena* modify the noun *saram* with the help of the postpositional particle *-n*.

The words can also modify nouns predicatively.

(78) a. *ge - ga eli - da*
he nom young dec
'He is young.'

b. *ge – ga ttena - da*
he nom leave dec
'He leaves.'

In sentences (77a) and (78a), what is the part of speech of the word *eli*, which modifies the noun? KorPar categorizes *eli* as a stative verb. Verbs can be classified as dynamic verbs or stative verbs. According to the definitions in Wikipedia (http://en.wikipedia.org/wiki/Stative_verb; http://en.wikipedia.org/wiki/Dynamic_verb), a dynamic verb is a verb that shows continued or progressive action on the part of the subject. In comparison, a stative verb is a verb that asserts that one of its arguments has a particular property. As mentioned in 5.1, the verbs in the lexicon have either a **dyn** or **sta** value for the feature **feat**. The morphological structure differs depending on these features and as a result, the dependency relation differs. According to Kim (2005), the dynamic verb can be attached to a present progressive tense marker, whereas the stative verb cannot.

(79) a. *ge -ga ttena -n -da*
he nom leave pres dec
'He is leaving.'

b. **ge -ga eli -n -da*
he nom young pres dec
'He is being young.' (ungrammatical sentence)

In (79), the verb *ttena* is a dynamic verb and can be attached to a present progressive tense marker, whereas *eli* is a stative verb, making it ungrammatical to be attached to the same tense marker. Thus, is a stative verb simply an adjective that cannot have any tense marker attached? No. Stative verbs can attach other tense markers, similar to dynamic verbs as in (80).

(80) a. *ge* *-nun* *yaet-nal* *-ey* *ttena* *-ss* *-da*
he nom long ago at leave pres dec
'He left a long time ago.'

b. *ge* *-nun* *yaet-nal* *-ey* *eli* *-ss* *-da*
he nom long ago at young pres dec
'He was young a long time ago.'

Therefore, both dynamic verbs and stative verbs can modify a noun attributively using the postpositional particle **vcn**. Stative verbs can also modify a noun predicatively, followed by the sentence-ending marker. KorPar categorizes words that can modify nouns both attributively (with the **vcn** particle attached) and predicatively as verbs.

There are, however, closed classes of words that modify a noun attributively without using **vcn**. These limited numbers of words are categorized as the **PartofSpeech**, **adj**, in the lexicon and include, for example, *iben* (this time), *ulma* (some amount), and *yeoro* (various).

5.4.5.1. PREDICATIVELY USED STATIVE VERBS

Stative verbs that modify nouns predicatively have a unique dependency structure allowing them to link two nominative case markers.

(81) *ge* ***-nun*** *noon* ***-i*** ***yaeppe*** *-da*
he **nom** eye **nom** **pretty** dec
'His eyes are pretty.'

In (81), the stative verb *yaeppe* is the head of two nominative markers, *nun* and *i*. Therefore, the rule for stative verbs linked with two nominative case markers is stated in (82).

(82) **% two nom pp/v**
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Y]],_,v,Z]) :- !, Z =feat:sta,
(X = case:nom ; X=case:nom-topicalization; X=case:nom-also),
(Y = case:nom; Y=case:nom-topicalization; X=case:nom-also),
N < NN, M < NN .

5.4.5.2. ATTRIBUTIVELY USED STATIVE VERBS

If either a dynamic verb or a stative verb is attached to a postpositional **vcn**, these verbs can modify a noun attributively, yet there is a difference in the syntactic dependency relation between the preceding words and the following verbs. In the case of dynamic verbs, there is a dependency relation with the preceding words. However, in the case of stative verbs, there are some sentences in which the preceding words do have a dependency relation and some that do not. This presents a problem for KorPar and, as a result, there are sentences that are parsed incorrectly.

(83) a. *ge* ***-nun*** *na* ***-ege*** *eli* *-n* *gang-a-ji* *-lul* ***ju*** *-ss* *-da*
he **nom** me **to** young vcn dog acc **give** past dec
'He gave a young dog (puppy) to me.'

b. ***aju*** ***eli*** *-n* *gang-a-ji* *-lul* *ju* *-ss* *-da*
very **young** vcn dog acc give past dec
'_ gave a very young dog (puppy).'

Sentence (83a) illustrates the case where the stative verb *eli* modifies the noun *gang-a-ji* attributively with the help of a **vcn**. The preceding nominative case marker *nun* and postpositional adverbial marker *ege* are dependents of the verb *ju*, not the closer verb *eli*. On the

other hand, (83b) shows the same stative verb *eli* modifying a noun attributively. In this case, the preceding adverb *aju* is a dependent of the verb *eli*, not the further verb *ju*.

There is no rule to capture the different dependency relations and KorPar will parse (83a) incorrectly by linking *nun* and *ege* with the verb *eli*. It is only the case that sentences with stative verbs such as (84) can be correctly parsed in any cases. To prevent crossing of arcs, KorPar uses the predicate **parse_node** of (67) instead of (68) in 5.3.3. Now, let's assume that the sentence (73a) in 5.4.2, repeated in (84), has the structure of:

A B stative verb **vcn n**

(84) *21 segi -ey sege saerob -un jishik -ul gajang manni changjoha - n -da*
21 century in world new(v) vcn knowledge acc mostly many invent(v) pres dec
'In the 21st century, the new worldwide knowledge is invented most actively.'

If there is a previous word (*B: sege*) that is not a dependent of the stative verb (*saerob*), which modifies the noun attributively, then the words (*A*: ey) that come before the previous word (*B: sege*) of the stative verb, will not be linked with the stative verb in order to have no crossing of arcs as the predicate (67) states.

A possible solution to the problem of linking stative verbs and the preceding words is to use probability. If the probability of a sentence such as that in (83a) is higher than that of (83b), then KorPar can have an additional "HeadList modification" case in the predicate **parse_node**.

(85) **parse_node([N,D,W,v|X],[Head|HeadList],[Head|NewHeadList]) :- member(feat:sta,X),**
parse_node([N,D,W,v|X],HeadList,NewHeadList).

The grammar rule in (85) states that if the current input word is a stative verb, it should be placed in the HeadList. In other words, if the current input word is a stative verb, it should not be linked with the previous words.

5.4.6. OPTIONAL CASE MARKERS

As mentioned in 5.2.2, in casual Korean speech, case markers are often omitted. This can present a problem for KorPar since the case markers are the heads of nouns they attach to and the verb stem subcategorizes case markers based on the grammar rules in KorPar.

It is very important to establish a boundary for the term "natural language." KorPar is designed to parse Korean natural language used in articles of the media or in textbooks. Therefore, KorPar parses sentences with an intransitive verb and a subject without a case marker (25a) or sentences with a transitive verb and an object without a case marker (25b). However, sentences with a transitive verb and a subject without a case marker (25c) or sentences with a transitive verb and no case markers (25d) can be produced and perceived in casual speech only if given plenty of context and KorPar does not parse these sentences.

5.4.7. PROBABILITY CASES

KorPar uses hand-coded heuristic rules to parse a given sentence. There are several troubleshooting cases that could not be solved with rules. Among them, I have pointed out a possible solution to homophonous words and unrecognized words by using probability method.

5.4.7.1. HOMOPHONOUS WORDS

Korean language has several cases of homophonous words (Kang, Park & Kwon 2001). There are two different approaches to address these words. One is to assume that the morphological tagger can differentiate one from the other(s) by the surrounding sounds, including whitespace. For example, the word "ㅇ|" can be a pronoun, a nominative case marker, or a copular verb with the same phonetic transcription [i]. In the lexicon, the pronoun is represented as *ii*, the case marker as *i*, and the copular verb as -*i*. In a similar approach, the morphological tagger, POSTAG, tags the three homophonous words with different tags. (http://isoft.postech.ac.kr/research/POMLIP/pomlip.html)

There is also an alternative approach to this problem. We can determine the probability of each part of speech and use the word with the highest probability first. Kim, Kwon and Yoon (1996) combined a rule-based approach with a statistical method in morphological disambiguation. First, the system tries to disambiguate words by applying rules. If it fails, the system disambiguates words based on the frequency of a morpheme in the corpus. The accuracy rate of a method based only on rules was 95.3%, and the rate increased to 97.1% when both rules and a statistical method were used.

5.4.7.2. UNRECOGNIZABLE WORDS

If the input word is not in the lexicon, the probability approach can be used. Considering that Korean is a head-final language and assuming that the average length of the 100 test sentences is 20 words, any word with a **Number** value lower than 15 has a high probability of being a noun, while any word with a **Number** value higher than 15 is most likely a verb.

Furthermore, an unrecognized word would normally be an open class part of speech rather than a finite closed class. The predicate **look_for_word** stated in (69) can be modified for unrecognized words, as stated in (86).

(86)
```
%To allow words that are not in the lexicon to be considered as a noun or verb.
look_for_word(Number,Word,Node) :- \+ word(Word,_), Number  <  15,
                                   Node = [Number, [] , Word , n |_].
look_for_word(Number,Word,Node) :- \+ word(Word,_), Number  >  15,
                                   Node = [Number ,[] , Word , v |_].
```

CHAPTER 6

RESULTS AND EVALUATIONS

The lexicon has approximately 703 words and the dependency rules are categorized into 23 cases based on the parts of speech listed in Table 1. KorPar has been tested with consecutive 100 sentences (more than 2000 words) taken from articles in the daily newspaper, Chosun Ilbo. The average length of the sentences is approximately 19 words.

Accuracy is evaluated by counting the number of correct links of head and dependent pairs in precision rate, recall rate, and calculating the F-score. (Jäarvinen & Tapanainen 1997, Goutte & Gaussier 2005).

Precision = [(# of received correct links) ÷ (# of received links)] × 100

Recall = [(# of received correct links) ÷ (# of desired links)] × 100

F-score = the harmonic mean of the precision and recall

The evaluation of the overall head and dependent pairs in the test sentences is shown is Table 4.

Table 4. Evaluation of the Overall Dependencies

Precision	Recall	F-score
97.6 %	95 %	96.3 %

The number of test sentences was small (100 sentences), therefore the overall accuracy rate is high. We can predict that if given more test sentences, the accuracy rates will decrease. However, the 100 test sentences are composed of almost all possible dependency structures in Korean, thus we can also predict that the accuracy rate will not decrease dramatically.

The performance of linking head and dependent pairs based on different troubleshooting cases in Korean are evaluated in Table 5.

Table 5. Evaluation of Different Dependencies

Dependency	**Precision**
Subcategorization	99.9 %
Long-Distance Dependency	98.9 %
Stative Verb & Dependents	79.7 %
Optional Case Marker	75 %
Unrecognized Words	25 %

The subcategorizations and long-distance dependencies are parsed with a very high precision rate. The precision rate for the optional case marker is low because there were only eight sentences where the accusative case marker was omitted, six of which were linked correctly. KorPar demonstrated difficulty linking the correct head for the dependents preceding the stative verbs that are used in relative clauses. In addition, the cases where probability was used for unrecognized words show poor precision rate, which seems unreliable.

The following examples are the input and output of the first five sentences in one of the tested articles.

?- try([hyundae-in,euy, jil-byung, un,eu-nenal,gap-jagi, balseng-dwe,-nun,geut,i,ani,da]).

12 [da, p, sem:declarative]
11 [ani, ccp, sem:not-be]
10 [i, pp, ending:con..case:nom..num:sing]
9 [geut, n, ending:con..sem:thing-or-fact]
8 [-nun, vcn, ending:vow..tense:pres..ending1:vow]
7 [balseng-dwe, v, ending:vow..sem:be-occured..subcat:1..feat:dyn]
6 [gap-jagi, adv, sem:suddenly]
5 [eu-nenal, adv, sem:one-day]
4 [un, pp, ending:con..case:nom-topicalization..num:sing]
3 [jil-byung, n, ending:con..sem:disease]
2 [euy, gen, ending:con..sem:of]
1 [hyundae-in, n, ending:con..sem:the-moderns]

Yes

‘The disease of the moderns does not occur suddenly in one day.’

?- try([imi,jen,bute, saeng-gi,-go-serv,it,ess,-deun,geut,-i,n-dae,daman,bon-in,i,molu,-go-serv,it,ess,da]).

19 [da, p, sem:declarative]
18 [ess, tp, ending:con..tense:past]
17 [it, v, ending:con..sem:be-or-exist..subcat:1..feat:dyn]
16 [-go-serv, serv, sem:serial-verb-connector]
15 [molu, v, ending:vow..sem:not-know..subcat:2..feat:dyn]
14 [i, pp, ending:con..case:nom..num:sing]
13 [bon-in, n, ending:con..sem:oneself]
12 [daman, adv, sem:simply-or-however]
11 [n-dae, vcv, ending:vow..sem:although-or-compared-to]
10 [-i, cp, sem:be]
9 [geut, n, ending:con..sem:thing-or-fact]
8 [-deun, vcn, tense:past]
7 [ess, tp, ending:con..tense:past]
6 [it, v, ending:con..sem:be-or-exist..subcat:1..feat:dyn]
5 [-go-serv, serv, sem:serial-verb-connector]
4 [saeng-gi, v, ending:vow..sem:come-into-existence..subcat:1..feat:dyn]
3 [bute, advp, ending:con..sem:since]
2 [jen, n, ending:con..sem:before-or-past-entire]
1 [imi, adv, sem:already]

Yes

‘It has already started to exist a long time ago, however, we were not aware of it.’

?- try([am,sae-po,nun,mae-il,uli,mom,eyse, balsaengha,-go,salaji,-e,-ga,n,da]).
14 [da, p, sem:declarative]
13 [n, tp, ending:vow..tense:pres]
12 [-ga, v, ending:vow..sem:go..subcat:1..feat:dyn]
11 [-e, serv, sem:serial-verb-connector]
10 [salaji, v, ending:vow..sem:disappear..subcat:1..feat:dyn]
9 [-go, vcv, ending:vow..sem:also-or-as-or-for]
8 [balsaengha, v, ending:vow..sem:occur..subcat:1..feat:dyn]
7 [eyse, advp, ending:con..sem:at-or-in-that]
6 [mom, n, ending:con..sem:body]
5 [uli, n, ending:vow..sem:us]
4 [mae-il, adv, sem:everyday]
3 [nun, pp, ending:vow..case:nom-topicalization..num:sing]
2 [sae-po, n, ending:vow..sem:cell]
1 [am, n, ending:con..sem:cancer]

Yes

‘The cancer cell appears and disappears in our body everyday.’

?- try([bal-am, gwa-jeng,un, shi-jak,gwa,whak-san, gwa-jeng,euro,nanu,-e, sel-myungha,l-su,it,da]).
14 [da, p, sem:declarative]
13 [it, v, ending:con..sem:be-or-exist..subcat:1..feat:dyn]
12 [l-su, vcv, ending:vow..sem:can]
11 [sel-myungha, v, ending:vow..sem:explain..subcat:2..feat:dyn]
10 [-e, serv, sem:serial-verb-connector]
9 [nanu, v, ending:vow..sem:divide..subcat:2..feat:dyn]
8 [euro, advp, ending:con..sem:to-or-as-or-with]
7 [gwa-jeng, n, ending:con..sem:process]
6 [whak-san, n, ending:con..sem:spreading]
5 [gwa, scs, ending:con..sem:and]
4 [shi-jak, n, ending:con..sem:start]
3 [un, pp, ending:con..case:nom-topicalization..num:sing]
2 [gwa-jeng, n, ending:con..sem:process]
1 [bal-am, n, ending:con..sem:carcinogenesis]

Yes

‘The carcinogensis process can be explained by dividing into intial starting stage and spreading stage.’

?- try([shi-jak, gwa-jeng,un,sae-po,euy, u-jenja,ey,sonsang,i, saeng-gi,e(se),sae-po,euy,seng-jil,i,jeng-sang,eyse,ii-talha,-nun, byunwha,lul, whi-miha,n,da]).
24 [da, p, sem:declarative]
23 [n, tp, ending:vow..tense:pres]
22 [whi-miha, v, ending:vow..sem:mean..subcat:2..feat:dyn]
21 [lul, pp, ending:vow..case:acc..num:sing]
20 [byunwha, n, ending:vow..sem:change]
19 [-nun, vcn, ending:vow..tense:pres..ending1:vow]
18 [ii-talha, v, ending:vow..sem:to-breakaway..subcat:2..feat:dyn]
17 [eyse, advp, ending:con..sem:at-or-in-that]
16 [jeng-sang, n, ending:con..sem:normality]
15 [i, pp, ending:con..case:nom..num:sing]
14 [seng-jil, n, ending:con..sem:temper-or-nature]
13 [euy, gen, ending:vow..sem:of]
12 [sae-po, n, ending:vow..sem:cell]
11 [e(se), vcv, ending:vow..sem:because-or-then]
10 [saeng-gi, v, ending:vow..sem:come-into-existence..subcat:1..feat:dyn]
9 [i, pp, ending:con..case:nom..num:sing..feat:sta]
8 [sonsang, n, ending:con..sem:damage]
7 [ey, advp, ending:vow..sem:of-or-to-or-on-or-at-or-in-or-in-addition-to..case:nom]
6 [u-jenja, n, ending:vow..sem:gene]
5 [euy, gen, ending:vow..sem:of]
4 [sae-po, n, ending:vow..sem:cell]
3 [un, pp, ending:con..case:nom-topicalization..num:sing]
2 [gwa-jeng, n, ending:con..sem:process]
1 [shi-jak, n, ending:con..sem:start]

Yes

‘The intial stage is a stage where the the cell gene gets damaged and the natures of the cell breakaway from normal cells.’

CHAPTER 7

CONCLUSION

KorPar is a rule-based parser for the partially free-word-order language, Korean. It is based on dependency grammar for efficiency and unification-based grammar for separation of the grammar rules and the algorithm itself. It is implemented non-deterministically in Prolog, and can therefore backtrack to any alternative in the lexicon, grammar rules, or algorithm section.

The input of the parser is a list of basic morphemes (simply stated as "word" in previous chapters) that have either a meaning or a function. The output is a dependency structure, represented with indentation and composed of information including case, number, tense, meaning, and so forth.

The overall accuracy rate of linking the correct dependency pairs resulted in an F-score of 96.3%, precision of 97.5% and recall of 95%, using sample sentences from a newspaper. It is often the case that a rule-based parser would have a high precision rate and a relatively low recall rate (Sagae, MacWhinney and Lavie 2004; Lim, Lee and Jang 2005). However, KorPar manages to maintain high rates for both. If the dependency pairs that are not indicated in the grammar rule were forced to be linked together, then the rate of recall would increase and that of precision would decrease.

KorPar has a very high precision rate in linking correct subcategorizations and detecting long-distance dependencies. However, its weakness is a low precision rate in linking the correct head(s) for the words preceding the stative verbs that modify nouns with the complementizer **vcn**.

In addition, although the precision rate is fairly low when using probability for unrecognized words, KorPar can easily add probability methods to other difficult structures that the rules could not parse, because the grammar and the algorithm itself are separated.

7.1. CONTRIBUTIONS OF KORPAR

KorPar is a rule-based parser for Korean based on dependency grammar. Unlike other dependency parsers, KorPar is implemented in Prolog for automatic backtracking and unification procedures. Although the number of test sentences is small, KorPar shows relatively high precision and recall rates. Since the test sentences include various possible dependency structures of Korean, this result provides a positive perspective of the performance of KorPar.

Korean is a partially free-word-order language and there are many cases of ambiguities in the decision of head and dependent pairs, particularly in long-distance dependency relations. KorPar shows a successful precision rate of 98.9% for these cases, simply by the grammar rules and the unique searching algorithm of the parser.

The dependency grammar represents partially free-word-order of Korean and facilitates the implementation of the parser. The unification-based grammar enables the grammar to identify head and dependent relations in a simple manner and separates the algorithm from the grammar for easy maintenance.

7.2. POSSIBLE FUTURE IMPROVEMENTS

KorPar can be improved by enlarging the lexicon and testing more sentences. In addition, the probability method can be used for other kinds of troubleshooting cases that the grammar

rules could not capture. For example, the dependency relation between the words that precede the stative verb in a relative clause can be parsed with a better precision rate by using statistics. This is especially true since there are cases where all of the preceding words are dependents of the stative verb, cases where none of them are dependents, and even worse cases where some of the preceding words are dependents and some are not. The probability rates of these three different cases can be examined and used for better performance.

A further possible step would entail applying KorPar to other free-word-order languages. This could be accomplished by simply substituting the lexicon and grammar rules sections without necessitating a change in the algorithm section.

REFERENCES

Abney, Steven P. (1989) A computational model of human parsing. *Journal of Psycholinguistic Research* 18:129-144.

Agel, V.; Eichinger, L.M.; Eroms, H.W.; Hellwig, Peter; Heringer, Hans Jürgen; and Lobin, Henning (2003) *Dependency and valency: an international handbook of contemporary research*. Portico Publications

Arnola, Harri and Oy, Kielikone (1998) On parsing binary dependency structures deterministically in linear time. *Workshop on dependency-based grammars, COLING-ACL*'98, *Montreal,* 68-77.

Bauer, L. (1979) Some thoughts on dependency grammar. *Linguistics* 17:301-315.

Barton, Berwick, and Ristad (1987) *Computational complexity and natural language*. MIT Press.

Beck, David (1999) *The typology of parts of speech system: the markedness of adjectives*. Doctoral dissertation,University of Toronto.

Cha, Jeongwon; Lee, Geunbae; and Lee, Jonghyeuk (2002) Korean combinatory categorial grammar and statistical parsing. *Computers and the Humanities* 36.4:431-453.

Charniak, Eugene (1997) Statisitical techniques for natural language parsing. *AI Magazine*, 18.4: 33-44.

Cho, Hyung Joon and Park, Jong (2000) Informed parsing for coordination with combinatory categorial grammar, *Proceedings of the International Conference on Computational Linguistics.* 593-599.

Choi, Hyen-Pay (1971). *Wuli Malpon* (‘Our Grammar’). Seoul: Jengumsa.

Choi, Seongsook (2003) Serial verbs and the empty category. *Proceedings of the workshop on Multi-Verb constructions*. Trondheim Summer School 2003.

Chomsky, Noam (1995) Bare Phrase Structure. In Gert Webelhuth, ed., *Government and Binding Theory and the Minimalist Program*.Oxford: Blackwell, pp. 383-439.

Chung, Hoojung and Rim, Hae-Chang (2004) *Unlexcialized dependency parser for variable word order languages based on local contextual pattern*. CICLing. Lecture note in Computer Science 2945:112-124.

Chung, Hoojung (2004) *Statistical Korean dependency parsing model based on the surface contextual information*. Ph.D. dissertation of Korea University.

Clark, Stephen; Hockenmaier , Julia; and Steedmean, Mark (2002) Building deep dependency structures with a wide-coverage CCG parser. *In Proceedings of the 40th Meeting of the ACL*. 327-334.

Covington, Michael A. (1990) Parsing discontinuous constituents in dependency grammar. *Computational Linguistics* 16:234-236.

Covington, Michael A. (1992) *GB Theory as Dependency Grammar*. Research Report AI-1992-03.

Covington, Michael A. (1994a) *GULP 3.1: An extension of Prolog for unification-based grammar*. Research Report AI-1994-06, Artificial Intelligence Programs, University of Georgia.

Covington, Michael A. (1994b) *Natural language processing for Prolog programmers*. Prentice-Hall, NJ.

Covington, Michael A. (2001) A fundamental algorithm for dependency parsing. *In Proceedings of 39th Annual ACM Southeast Conference.*

Covington, Michael A. (2003) *A free-word-order dependency parser in Prolog*. University of Georgia. http://www.ai.uga.edu/mc/ProNTo.

Germann, Ulrich (1999) A deterministic dependency parser for Japanese. *In Proceedings of the MT Summit VII*. Singapore.

Goutte, Cyril and Gaussier , Eric (2005) A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. *ECIR 27th European Conference on Information Retrieval*, Santiago de Compostela, Spain, 21-23 March.

Han, Chung-hye; Yoon, Juntae; Kim, Nari; and Mee-sook Kim (2000) A feature-based lexicalized tree-adjoining grammar for Korean. *IRCS* 4.

Hays, D. (1964) Dependency theory: a formalism and some observations. *Language* 40.4:511-525.

Hudson, R.A. (1980) Constituency and dependency. *Linguistics* 18:179-198.

Hudson, R.A. (1984) *Word grammar*. Basil Blackwell.

Hudson, R.A. (1993) Do we have heads in our minds? In Corbett, Greenville; McGlashen, Scott; Fraser, Norman, eds., *Heads in Grammatical Theory*, 266-291. Cambridge University Press.

Jackendoff, Ray (1977) *X-bar syntax: a study of phrase structure*. The MIT Press.

Jackendoff, Ray (2002) *Foundations of Language: Brain, Meaning, Grammar, Evolution.* Oxford University Press.

Jäarvinen, Timo and Tapanainen, Pasi (1997) A non-projective dependency parser. *In Proceedings of 5th Conference on Applied Natural Language Processing, ANLP-1995*, Washington, D.C. 64-71

Jäarvinen, Timo and Tapanainen, Pasi (1998) *Towards an implementable dependency grammar. Processing of Dependency-Based Grammars*, Kahane and Polguere, eds., pp.1-10.

Kang, Mi-young; Park, Su-Ho; and Kwon, Hyuk-chul (2001) A pattern analysis of lexical ambiguities and the Korean grammar checker with partial parsing. *In Proceedings of CKITOP* 2001. pp.45-52.

Kang, Mi-young; Park, Su-ho; Yoon, Ae-sun; and Kown, Hyuk-chul (2002) Potential governing relationship and a Korean grammar checker using partial parsing. *IEA/AIE*, pp.692-702.

Kang, Seungman (1997) A comparative analysis of SVCs and Korean V-V Compounds. The 40th Anniversary of Generativism. *Proceedings of electronic conference* Dec pp.1-12.

Kim, Changhyun; Kim, Jae-hoon; Seo, Jungyun; and Kim, Gil Chang (1994) A right-to-left chart parsing with headable paths for Korean dependency grammar. *Computer Processing of Chinese and Oriental Languages* 8:105-118.

Kim, Minjoo (2005) *The absence of adjective category in Korean*. Under revision for resubmission for Journal of East Asian Linguistics.

Kim, Min-jung; Kwon, Hyuk-chul; and Yoon, Ae-sun (1996) Rule-based approach to Korean morphological disambiguation supported by statistical method. *In Proceedings of the Eleventh PACLIC*: 237-246.

Kim, Mi-Young; Kang, Sin-Jae; and Lee, Jong-Hyeok (2001) Resolving ambiguity in inter-chunk dependency parsing. *Natural Language Processing Pacific Rim Symposium* pp.263-270.

Kim, Seongyong and Choi,Key-sun (2003) Automatic generation of composite labels using part-of-speech tags for parsing Korean. *International Journal of Computer Processing of Oriental Languages* 16.3:197-218.

Kim, Yeon-Jun; Byeon, Heo-Jin; and Oh, Yung-Hwan (1999). Prosodic phrasing in Korean; determine governor, and then split or not. *In Proceedings of Eurospeech* '99, Budapest.pp. 539-542.

Kornai and Pullum (1990) The X-bar theory of phrase structure. *Language* 66.1

Kruijff, Geert-Jan M. (2002) *Formal and computational aspects of dependency grammar*. From ESSLLI Summer School 2002.

Kwon, Hyuk-chul and Yoon, Ae-sun (1991) Unification-based dependency parsing of governor-final languages. *In Proccedings of IWPT*. pp.172-192.

Kwon, Hyuk-chul; Yoon, Aesun; and Kim, Yunk-taek (1990) A Korean analysis system based on unification and chart. *In Proceedings of Pacific Rim International Conference on Artificial Intelligence '90*.pp. 251-256.

Lee, Geunbae and Lee, Jong-Hyeuk (1995) SKOPE: A connectionist/symbolic architecture of spoken Korean processing. *Learning for Natural Language Processing*. pp.102-116.

Lee, Geunbae; Lee, Jong-Hyeuk; and Yoo, Jinhee (1997) Multi-level post-processing for Korean character recognition using morphological analysis and linguistic evaluation. *Pattern Recognition* 30.7:1347-1360.

Lee, Ikseb (2005) *Korean Grammar*. Seoul National University. (In Korean)

Lee, Kihwang (2003) *Yet Another Statistical Case Assignment in Korean*. Student Workshop, North American Summer School of Logic, Language and Computation (NASSLLI '03). 17-21 Jun, Bloomington (IN), USA.

Lee, Kong Joo; Kim, Jae-Hoon; and Kim, Gil Chang (1997) An efficient parsing of Korean sentences using restricted phrase structure grammar. *International Journal of Computer Processing of Oriental Languages* 11.1:49-61.

Lee, S. (2002) *A statistical model for identifying grammatical relations in Korean sentences.* Ph.D. thesis, Dept. of Computer Science. Sogang University.

Lee, Seungmi and Choi, Key-Sun (1997) Reestimation and best-first parsing algorithm for probablistic dependency grammars. *In Proceedings of the Fifth Workshop on Very Large Corpora*. pp41-55.

Lee, Won-il; Lee, Geunbae; and Lee, Jong-hyeuk (1995) Chart-driven connectionist categorial parsing of spoken Korean. *In Proceedings of the ICCPOL95* (Hawaii).

Lim, Soojong; Lee, Changki; and Jang, Myoungkil (2005) Restoring an Elided Entry Word in a Sentence for Encyclopedia QA System. *Second International Joint Conference on Natural Language Processing*

Mel'čuk, I. (1988) *Dependency Syntax: Theory and Practice.* SUNY Press, Albany NY.

Mel'čuk, I. (2003) Levels of dependency in linguistic description: concepts and problems. In *dependency and valency: an international handbook of contemporary research*. Portico Publications

Nam, HyeonSook; So, Kilja; Kim, SuNam; and Kwon, Hyuk-chul (1998) Resolving lexical and syntactic ambiguities using heuristic rules for Korean grammar checker. *Proceedings of Association for Intelligent Machinery*.3: 454-457.

Noord, Gertjan van (1993) *Reversibility in Natural Language Processing*. PhD thesis, University of Utrecht.

Park, So-young; Kwak, Yong-jae; and Rim, Hae-chang (2005) Feature-based Korean grammar utilizing learned constraint rules. *Computational Intelligence* 21.1:69-89.

Radford, Andrew (1997) *Syntactic theory and the structure of English: A minimalist approach* . Cambridge: Cambridge University Press.

Rayner, Manny; Boullion Pierrette; Hockey, Beth A.; Chatzichresafis, Nikos; and Starlander, Marianne (2004) Comparing rule-based and statistical approaches to speech understanding in a limited domain speech translation system. *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, Baltimore, MD.

Richardson, Stephen D. (1994) *Bootstrapping statistical processing into a rule-based natural language parser.* Submitted to ACL-94, February.

Sagae, K.; MacWhinney, B.; and Lavie, A. (2004) Adding syntactic annotations to transcripts of parentchild dialogs. In *LREC 2004* pp. 1815-1818. Lisbon:LREC.

Samuelsson , Christer. (2000) A statistical theory of dependency syntax. In *Proceedings of the 18th International Conference on Computational Linguistics.* pp.684-690.

Schneider, Gerold. (2003) Extracting and using trace-free functional dependencies from the Penn Treebank to reduce parsing complexity. In *Proceedings of Treebanks and Linguistic Theories*.

Schneider, Gerold. (1998) *A linguistic comparison of consitituency, dependency and link grammars*. Lizentiatsarbeit, Institut für Informatik der Zürich. http://www.ifi.unizh.ch/cl/study/lizarbeiten/lizgerold.pdf.

Shieber, S. M. (1986) *An introduction to unification-based approaches to grammar*. (CSLI Lecture Notes, 4.) Stanford: Center for the Study of Language and Information (distributed by University of Chicago Press).

Sohn, H. (1999) *The Korean language*. Cambridge University Press.

Steedman, M. (1993) Categorial Grammar (tutorial overview). *Lingua*. 90:221-258.

Stewart, Osamuyimen T. (2001) *The Serial Verb Construction Parameter*. (Outstanding dissertations in linguistics). Garland Publishing Inc.

Venable, Peter. (2001) Lynx: building a statistical parser from a rule-based parser. *In Proceedings of NAACL.*

Voll, Kimberly D.; Yeh, Tom P.; and Dahl, Veronica (2001) An assumptive logic programming methodology for parsing. *International Journal of Artificial Intelligence* 10.4:573-588.

Yoon, Juntae (2002) Efficient semi-deterministic parsing for Korean using lexical co-occurrence data from a corpus. *International Journal of Computer Processing of Oriental Languages* 15.4:493-516.

Yoon, J.;C.H. Han; N. Kim; and M. Kim (2000). Customizing the XTAG system for efficient grammar development for Korean. *In Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pp.221–226.

Yoon, J., C. Lee, S. Kim, and M. Song (1999) "Morphological Analyzer of Yonsei Univ., Morany: Morphological analysis based on large lexical database extracted from corpus. *In Proceedings of Korean Language Information Processing*. (In Korean)

Zeman, Daniel. 2002. How to decrease the performance of a statistical parser: on the negative influence on some outside factors. The Prague Bulletin of Mathematical Linguistics 78. 53-62.

APPENDICES

A. PROLOG CODE OF KORPAR

```
% korean parser KorPAR: Soyoung Kwon
% This is a dependency parser for Korean that uses unification-based gramamar
% last modified: May 24 2006

:- ensure_loaded('c://Program Files//pl//bin//GULP3swi-old.pl').

%%lexicon word(phonetic sound, partofspeech(features)).
%% n(noun)/pp(subject & object case marker)/pro(pronoun)/advp(adverbial post
marker)/v(verb)/p(sentence ending marker)/tp(tense post marker)/cp(copular post marker)/ccp(other kind of
copular marker)/adj(adjective)/gen(genitive)/num(numeral)/ncv(complementizer:noun dep& verb
head)/scs(complementizer:same kind of dep&head)/pcv(complementizer:p dep& verb
head)/vcv(complementizer:v dep&head)/adv(adverb)/vadjp(verb to adj post marker)/vnp(verb to noun
marker)/nnp(noun to noun marker)/serv(serial verb marker)/qm(quotational marker)


%% n(noun)
word(a-be-ji,n(ending:vow..sem:father)).
word(am,n(ending:con..sem:cancer)).
word(baekje,n(ending:vow..sem:baekje)).
word(baksa,n(ending:vow..sem:doctor)).
word(bal-am,n(ending:con..sem:carcinogenesis)).
word(baldal,n(ending:con..sem:growth-or-progress-or-advancement)).
word(bal-eum,n(ending:con..sem:pronunciation)).
word(bal-seng,n(ending:con..sem:occurence-or-outbreak)).
word(bande,n(ending:vow..sem:opposite)).
word(bang-e,n(ending:vow..sem:defense)).
word(bang-il,n(ending:con..sem:visit-to-japan)).
word(ban-young,n(ending:con..sem:reflection)).
word(bel-lae,n(ending:vow..sem:insect)).
word(bi-sok-e,n(ending:vow..sem:slang)).
word(bitamin,n(ending:con..sem:vitamin)).
word(bo-geup,n(ending:con..sem:popularization)).
word(bon-in,n(ending:con..sem:oneself)).
word(bu-chin,n(ending:con..sem:father)).
word(bujok,n(ending:con..sem:insufficiency)).
word(bun,n(ending:con..sem:person-honorific)).
word(bun-rang,n(ending:con..sem:amount)).
word(byung,n(ending:con..sem:disease)).
word(byunwha,n(ending:vow..sem:change)).
word(chae-so,n(ending:vow..sem:vegetable)).
word(chingu,n(ending:vow..sem:friend)).
word(cho-dung-hakgyo,n(ending:vow..sem:elementary-school)).
word(chu-kikyung,n(ending:con..sem:cardinal)).
word(dae,n(ending:vow..sem:when-or-at)).
```

word(dae-bubun,n(ending:con..sem:majority)).
word(daehak-kyo,n(ending:vow..sem:university)).
word(daejung-wha,n(ending:vow..sem:massification)).
word(dang-bun,n(ending:con..sem:amount-of-sugar)).
word(dae-wha,n(ending:vow..sem:conversation)).
word(dan-e,n(ending:vow..sem:vocabulary)).
word(ddae,n(ending:vow..sem:when)).
word(ddaemun,n(ending:con..sem:reason)).
word(ddal,n(ending:con..sem:daughter)).
word(ddet,n(ending:con..sem:meaning-or-philosophy)).
word(deng-e-li,n(ending:vow..sem:lump-or-mass)).
word(dna,n(ending:vow..sem:dna)).
word(dogu,n(ending:vow..sem:tool)).
word(dong-mul,n(ending:con..sem:animal)).
word(dwen-sori,n(ending:vow..sem:strong-sounds)).
word(e-linhi,n(ending:vow..sem:child)).
word(em-ma,n(ending:vow..sem:mom)).
word(enoh-hak,n(ending:con..sem:linguistics)).
word(eu-je,n(ending:vow..sem:yesterday)).
word(eun-e,n(ending:vow..sem:jargon)).
word(eungu,n(ending:vow..sem:research)).
word(euy-mi,n(ending:vow..sem:meaning)).
word(gae-tong,n(ending:con..sem:system)).
word(gaji,n(ending:vow..sem:type)).
word(gaji,n(ending:vow..sem:kind)).
word(gal-geun,n(ending:con..sem:no)).
word(gap,n(ending:con..sem:price)).
word(gang-a-gi,n(ending:vow..sem:dog)).
word(gan-pan,n(ending:con..sem:signboard)).
word(gaseum,n(ending:con..sem:heart)).
word(gel,n(ending:con..sem:character)).
word(gengang,n(ending:con..sem:health)).
word(geut,n(ending:con..sem:thing-or-fact)).
word(ggwa,n(ending:vow..sem:department)).
word(gisul,n(ending:con..sem:skill)).
word(giwhoo,n(ending:vow..sem:climate)).
word(gong-bu,n(ending:vow..sem:study)).
word(got,n(ending:con..sem:place)).
word(go-u-e,n(ending:vow..sem:pure-native-language)).
word(go-yang-i,n(ending:vow..sem:cat)).
word(gubyul,n(ending:con..sem:distinction)).
word(gukne,n(ending:vow..sem:domestic)).
word(gwahak,n(ending:con..sem:science)).
word(gwa-il,n(ending:con..sem:fruit)).
word(gwa-jeng,n(ending:con..sem:process)).
word(gwa-mok,n(ending:con..sem:subject)).
word(gwang-sen,n(ending:con..sem:beam-or-light)).
word(gwang-woo-byung,n(ending:con..sem:mad-cow-disease)).
word(gyosoo,n(ending:vow..sem:professor)).
word(gyue-jae,n(ending:vow..sem:textbook)).
word(haek,n(ending:con..sem:nucleas)).
……..
……..
word(yeul,n(ending:con..sem:heat)).
word(yeum-jung,n(ending:con..sem:aversion)).
word(yeun-guso,n(ending:vow..sem:research-center)).

```
word(yeun-gwan,n(ending:con..sem:connection-or-relation)).
word(yoin,n(ending:con..sem:main-cause)).
word(pero,n(sem:percentage)).


%% pro (pronoun)
word(ge,pro(ending:vow..sem:he-or-she-or-him-or-her-or-the)).
word(je,pro(ending:vow..sem:that)).
word(ii,pro(ending:vow..sem:this-or-it)).


%% pp(subject & object case marker)
word(ga,pp(case:nom..ending:vow..num:sing)).
word(i,pp(case:nom..ending:con..num:sing)).
word(lul,pp(case:acc..ending:vow..num:sing)).
word(ul,pp(case:acc..ending:con..num:sing)).
word(nun,pp(case:nom-topicalization..ending:vow..num:sing)).
word(un,pp(case:nom-topicalization..ending:con..num:sing)).
word(do,pp(case:nom-also..num:sing)).
word(lan,pp(case:nom-meaning..ending:vow..num:sing)).
word(i-lan,pp(case:nom-meaning..ending:con..num:sing)).


%% plp(plural marker)
word(dul,plp(num:pl..ending:con)).


%%advp(adverbial post marker)
word(euro,advp(ending:con..sem:to-or-as-or-with)).
word(ro,advp(ending:vow..sem:to-or-as)).
word(ey,advp(sem:of-or-to-or-on-or-at-or-in-or-in-addition-to)). %%% ending of the noun does not matter.
word(eyse,advp(sem:at-or-in-that)).
word(ggaji,advp(sem:uptil)).
word(igido,advp(sem:might-also-be)).
word(chelem,advp(sem:like)).
word(ege,advp(sem:to)).
word(mada,advp(sem:every)).
word(ddae,advp(sem:when)).
word(bute,advp(sem:since)).
word(man-keum,advp(sem:as-much)).
word(dul-ro,advp(sem:with)).
word(ey-euyhe,advp(sem:by)).
word(jung,advp(among-or-during)).
word(gwa-hamkkae,advp(ending:con..sem:together)).
word(wa-hamkkae,advp(ending:vow..sem:together)).


%% v(verb)
word(al,v(ending:con..subcat:2..sem:know..feat:dyn)).
word(alum-dab,v(ending:con..subcat:1..sem:be-beautiful..feat:sta)).
word(alyu-ji,v(ending:vow..subcat:1..sem:become-known..feat:dyn)).
word(anta-ggab,v(ending:con..subcat:1..sem:pity..feat:dyn)).
word(appdu,v(ending:vow..sucat:2..sem:have-ahead..feat:dyn)).
word(appseu,v(ending:vow..subcat:2..sem:before..feat:dyn)).
word(bae-u,v(ending:vow..subcat:2..sem:learn..feat:dyn)).
```

```
word(baggu,v(ending:vow..subcat:2..sem:change..feat:dyn)).
word(baggwui-e-ga,v(ending:vow..subcat:1..sem:be-changed..feat:dyn)).
word(bala,v(ending:vow..subcat:2..sem:wish..feat:dyn)).
word(balki,v(ending:vow..subcat:2..sem:lighten-or-express..feat:dyn)).
word(balpyoha,v(ending:vow..subcat:2..sem:announce..feat:dyn)).
word(balsaengha,v(ending:vow..subcat:1..sem:occur..feat:dyn)).
word(balseng-dwe,v(ending:vow..subcat:1..sem:be-occured..feat:dyn)).
word(balumha,v(ending:vow..subcat:2..sem:pronounce..feat:dyn)).
word(battel,v(ending:con..subcat:2..sem:follow..feat:dyn)).
word(bigyu-dwe,v(ending:vow..subcat:2..sem:be-compared..feat:dyn)).
word(bo-i,v(ending:vow..subcat:2..sem:to-show..feat:dyn)).
word(bokjab-heji,v(ending:vow..subcat:1..sem:become-complicated..feat:dyn)).
word(bone,v(ending:vow..subcat:2..sem:send..feat:dyn)).
word(bulgwaha,v(ending:vow..subcat:1..sem:no-more-than..feat:dyn)).
word(bul-pyenha,v(ending:vow..subcat:1..sem:be-uncomfortable..feat:sta)).
word(bun-yelha,v(ending:vow..subcat:2..sem:breakup-or-division..feat:dyn)).
word(byunha,v(ending:vow..subcat:1..sem:change..feat:dyn)).
word(chajabo,v(ending:vow..subcat:2..sem:try-to-find..feat:dyn)).
word(changjoha,v(ending:vow..subcat:2..sem:create..feat:dyn)).
word(chot,v(ending:con..subcat:2..sem:chase..feat:dyn)).
word(chuguha,v(ending:vow..subcat:2..sem:seek..feat:dyn)).
word(daechi,v(ending:vow..subcat:2..sem:parboil..feat:dyn)).
word(dala-ju,v(ending:vow..subcat:2..sem:put-on-or-hang..feat:dyn)).
word(dale,v(ending:vow..subcat:1..sem:be-different..feat:sta)).
word(dal-laji,v(ending:vow..subcat:1..sem:become-different..feat:sta)).
word(danhoha,v(ending:vow..subcat:1..sem:firm-or-decisive..feat:dyn)).
word(dayangha,v(ending:vow..subcat:1..sem:diverse..feat:sta)).
word(ddae-muni,v(ending:vow..subcat:1..sem:because-of..feat:dyn)).
word(ddale,v(ending:vow..subcat:1..sem:follow..feat:dyn)).
word(ddalu,v(ending:vow..subcat:2..sem:follow..feat:dyn)).
word(dde-geb,v(ending:con..subcat:1..sem:be-hot..feat:sta)).
……..
……..
word(silchenha,v(ending:vow..subcat:2..sem:practice..feat:dyn)).
word(silhem-hebo,v(ending:vow..subcat:2..sem:try-out..feat:dyn)).
word(ssa,v(ending:vow..subcat:1..sem:be-cheap..feat:sta)).
word(sse,v(ending:vow..subcat:2..sem:use..feat:dyn)).
word(tanseng-siki,v(ending:vow..subcat:2..sem:give-birth-or-bring-about..feat:dyn)).
word(tek-byelha,v(ending:vow..subcat:1..sem:be-special..feat:sta)).
word(un-youngha,v(ending:vow..subcat:2..sem:manage..feat:dyn)).
word(whe-woo,v(ending:vow..subcat:2..sem:memorize..feat:dyn)).
word(whi-miha,v(ending:vow..subcat:2..sem:mean..feat:dyn)).
word(wiha,v(ending:vow..subcat:2..sem:value-or-serve..feat:dyn)).
word(yae-chuk-ha,v(ending:vow..subcat:2..sem:predict..feat:dyn)).
word(yaeppe,v(ending:vow..subcat:1..sem:be-pretty..feat:sta)).
word(yeli,v(ending:vow..subcat:1..sem:be-opened..feat:dyn)).
word(yeppe,v(ending:vow..subcat:1..sem:be-pretty..feat:sta)).
word(yuriha,v(ending:vow..subcat:1..sem:be-in-advantage-of..feat:dyn)).


%% p(sentence ending)
word(da,p(sem:declarative)).


%% tp(tense ending)
word(nun,tp(tense:pres..ending:con)).
```

```
word(n,tp(tense:pres..ending:vow)).
word(go-iss,tp(tense:pres-prog)).
word(go-iss-b-ni,tp(tense:pres-prog-honorific)).
word(b-ni,tp(tense:pres-honorific..ending:vow)).
word(seb-ni,tp(tense:pres-honorific..ending:con)).
word(ess,tp(tense:past..ending:con)).
word(ss,tp(tense:past..ending:vow)).
word(ess-b-ni,tp(tense:past-honorific..ending:con)).
word(ss-b-ni,tp(tense:past-honorific..ending:vow)).
word(l-geut-i,tp(tense:future..ending:vow)).
word(ul-geut-i,tp(tense:future..ending:con)).
word(gaet,tp(tense:future-will)).

%% serv (serial verb)
word(-e,serv(sem:serial-verb-connector)).
word(-go-serv,serv(sem:serial-verb-connector)).


%% cp(copular postmarkers)
word(-i,cp(sem:be)).


%% ccp(other kind of copular verb)
word(dwe,ccp(sem:become)).
word(ani,ccp(sem:not-be)).


%% adj(adjective)
word(iben,adj(sem:this-time)).
word(ulma,adj(sem:some-amount)).
word(yeoro,adj(sem:various)).
word(saege-jek,adj(sem:global)).
word(moden,adj(sem:every)).
word(changjo-jek,adj(sem:creative)).
word(on-gat,adj(sem:various)).
word(on,adj(sem:entire)).
word(musun,adj(sem:some)).


%% gen(genitive)
word(euy,gen(sem:of)).
word(la-nun,gen(ending:vow..sem:so-called)).
word(i-la-nun,gen(ending:con..sem:so-called)).

%% num(numeral)
word(myut,num(sem:several)).
word(13,num(sem:thirteen)).
word(21,num(sem:twenty-one)).
word(7,num(sem:seven)).
word(han,num(sem:one)).
word(30,num(sem:thirty)).
word(1,num(sem:one)).
word(du,num(sem:two)).
word(5,num(sem:five)).
word(10,num(sem:ten)).
```

%%ncv(complementizer that follows a noun and heads verb)
word(edo,ncv(subcat:1..sem:also)).
word(edo-bulgu-hago,ncv(subcat:1..sem:although)).
word(la-go,ncv(subcat:1..ending:vow..sem:says-that)).
word(i-la-go,ncv(subcat:1..ending:con..sem:says-that)).
word(boda,ncv(subcat:1..sem:compared-to)).
word(boda-nun,ncv(subcat:1..sem:compared-to)).
word(wehese,ncv(subcat:1..sem:for)).
word(e-bihe,ncv(subcat:1..sem:compared-to)).
word(mada,ncv(subcat:1..sem:each)).

%% scs(complementizer that taked the same part of speech(n or v or cp) as head & dep
word(ha-go,scs(sem:and)).
word(gwa,scs(ending:con..sem:and)).
word(wa,scs(ending:vow..sem:and)).
word(a-nila,scs(sem:not)).
word(i-na,scs(ending:con..sem:or)).
word(na,scs(ending:vow..sem:or)).
word(gat-eun,scs(sem:like-or-such-as)).
word(mit,scs(sem:as-well-as)).

%% pcv(complementizer that follows a sentence ender p)
word(go,pcv(sem:that)).
word(he-do,pcv(sem:although)).

%% vcv(complementizer that follows a verb & takes v as head)
word(a(se),vcv(sem:because-or-then)).
word(a-ya,vcv(ending:con..sem:conditional)).
word(do-rok,vcv(sem:in-order-to)).
word(e(se),vcv(sem:because-or-then)).
word(eu-myue,vcv(ending:con..sem:also)).
word(eu-myun,vcv(ending:con..sem:if)).
word(eu-myun-se,vcv(ending:con..sem:as-or-whole)).
word(eu-ryue-myun,vcv(ending:con..sem:if)).
……..
……..
word(n-dae,vcv(ending:vow..sem:although-or-compared-to)).
word(nun-dae,vcv(ending:con..sem:although-or-compared-to)).
word(nun-ji,vcv(sem:if)).
word(ryue-myun,vcv(ending:vow..sem:if)). %'a-ya' or 'ya'
word(ul-su,vcv(ending:con..sem:can)).
word(ul-surok,vcv(ending:con..sem:the-more)).
word(ya,vcv(ending:vow..sem:conditional)).

%% vcn (complementizers that take a verb as dep. and noun as head)
word(-n,vcn(ending:vow..tense:past)).
word(-un,vcn(ending:con..tense:past)).
word(-nun,vcn(ending:vow..tense:pres)).
word(-nun,vcn(ending:con..tense:pres)).%%%%%%%%%nun different!!!!!!!
word(-l,vcn(ending:vow..tense:future)).
word(-ul,vcn(ending:con..tense:future)).

```
word(-l- manhan,vcn(ending:vow..sem:similar-to)).
word(-ul- manhan,vcn(ending:con..sem:similar-to)).
word(-deun,vcn(tense:past)).


%%adv(adverb)
word(bballi,adv(sem:quickly)).
word(bel-ddek-bel-ddek,adv(sem:quickly)).
word(daa,adv(sem:all)).
word(daman,adv(sem:simply-or-however)).
word(de,adv(sem:more)).
word(debul-e,adv(sem:with-or-together)).
word(ddalase,adv(sem:therefore)).
……..
……..
word(sero,adv(sem:newly)).
word(sil-je-ro,adv(sem:in-reality)).
word(sashil,adv(sem:in-reality)).
word(son-shipgae,adv(sem:easily)).
word(teuk-hi,adv(sem:especially)).
word(tong-jje-ro,adv(sem:as-a-whole)).
word(whel-shin,adv(sem:a-lot)).
word(wi-he-se,adv(sem:for)).
word(yak-kan,adv(sem:a-little)).
word(yek-shi,adv(also)).


%%tadv(tense adverb)
word(euje-adv,tadv(tense:past..sem:yesterday)).
word(naeil-adv,tadv(tense:future..sem:tomorrow)).
word(geum-bang,tadv(tense:past..sem:a-while-ago)).
word(goud,tadv(tense:future..sem:soon)).
word(ap-euro,tadv(tense:future..sem:in-the-future)).


%% vnp(verbs to nouns)
word(gi,vnp(ending2:vow)).
word(eum,vnp(ending1:con..ending2:con)).
word(m,vnp(ending1:vow..ending2:con)).


%% nnp(noun to noun)
word(wha,nnp(sem:conversion-or-ization)).

%% quotation mark
word(-qm,qm(sem:quotationmark)).



%% dependency rule : the order in a sentence needs to be considered.
%% each word is represented as,
%% [NumberInSentence,DependentList,PhoneticSound,PartofSpeech,GULPfeatures]

/**************NOUN*************************************************************/
% noun/postpositional marker(pp)
check_dh([N,_,_,n,X],[NN,_,_,pp,Y]) :-  !, X = ending:End, Y = ending:End,NN is N +1.
```

```
% n/n
check_dh([N,_,_,n,X],[NN,_,_,n,Y]) :-  !, NN > N.

% num(numeral)/n
check_dh([N,_,_,num,X],[NN,_,_,n,Y]) :-  !, NN > N.

/**************QM*********************************************************/
% anything/qm
check_dh([N,_,_,_,_],[NN,_,_,qm,_]) :- !, NN is N + 1.

% qm/pp
check_dh([N,_,_,qm,_],[NN,_,_,pp,_]) :- !, NN is N + 1.

/**************PRO********************************************************/
%adj/pro
check_dh([N,_,_,adj,X],[NN,_,_,pro,Y]) :-  !, NN is N +1.

%pro/pp
check_dh([N,_,_,pro,X],[NN,_,_,Z,Y]) :-  !, (Z = pp; Z = n; Z = advp; Z = gen; Z = ncv; Z = scs), NN is N +1.

%pro/n
check_dh([N,_,_,pro,_],[NN,_,_,n,_]) :- !, NN > N.

/**********************PLP*********************************************/
%n/plp
check_dh([N,_,_,n,X],[NN,_,_,plp,Y]) :-  !, NN is N +1.

%plp/pp
check_dh([N,_,_,plp,X],[NN,_,_,pp,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.

%plp/advp
check_dh([N,_,_,plp,X],[NN,_,_,advp,Y]) :-  !, NN is N +1.

/***************NCV*****************************************************/
% ncv(complementizer)/v
check_dh([N,_,_,ncv,X],[NN,_,_,v,Y]) :-  !, NN > N.

%noun/ncv
check_dh([N,_,_,n,X],[NN,_,_,ncv,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,_],[NN,_,_,ncv,_]) :-  !, NN is N +1.

/***************SCS*****************************************************/
% noun/scs
check_dh([N,_,_,n,X],[NN,_,_,scs,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,_],[NN,_,_,scs,_]) :-  !, NN is N +1.

%verb/scs
check_dh([N,_,_,v,_],[NN,_,_,scs,_]) :-  !, NN is N +1.

%cp/scs
check_dh([N,_,_,cp,_],[NN,_,_,scs,_]) :-  !, NN is N +1.

%scs/noun
check_dh([N,[M,_,_,n,_]],_,scs,_],[NN,_,_,n,_]) :- !, N is M + 1, N < NN.
```

```
%scs/verb
check_dh([N,[[M,_,_,v,_]],_,scs,_],[NN,_,_,v,_]) :- !, N is M + 1, N < NN.

%scs/cp
check_dh([N,[[M,_,_,cp,_]],_,scs,_],[NN,_,_,cp,_]) :- !, N is M + 1, N < NN.

%vnp/scs
check_dh([N,_,_,vnp,_],[NN,_,_,scs,_]) :-  !, NN is N +1.

%scs/vnp
check_dh([N,[[M,_,_,vnp,_]],_,scs,_],[NN,_,_,vnp,_]) :- !, N is M + 1, N < NN.

/******************PCV************************************************/
% pcv(complementizer)/v
check_dh([N,_,_,pcv,X],[NN,_,_,v,Y]) :-  !, NN is N +1.

% p/pcv(complementizer)
check_dh([N,_,_,p,X],[NN,_,_,pcv,Y]) :-  !, NN is N +1.

/******************VCV************************************************/
%v/vcv
check_dh([N,_,_,v,X],[NN,_,_,vcv,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,v,_],[NN,_,_,vcv,_]) :-  !, NN is N +1.

% cp/vcv(complementizer)
check_dh([N,_,_,cp,X],[NN,_,_,vcv,Y]) :-  !, NN is N +1.

% ccp/vcv(complementizer)
check_dh([N,_,_,ccp,X],[NN,_,_,vcv,Y]) :-  !, NN is N +1.

% vcv(complementizer)/v
check_dh([N,_,_,vcv,X],[NN,_,_,v,Y]) :-  !, NN > N.

% vcv(complementizer)/cp
check_dh([N,_,_,vcv,X],[NN,_,_,cp,Y]) :-  !, NN > N.

% vcv(complementizer)/ccp
check_dh([N,_,_,vcv,X],[NN,_,_,ccp,Y]) :-  !, NN > N.

/***************VCN************************************************/
%tp/deun
check_dh([N,_,_,X,_],[NN,_,-deun,vcn,Y]):- !, (X = v; X = tp), NN is N +1.

%verb/vcn
check_dh([N,_,_,v,X],[NN,_,_,vcn,Y]):- !, X = ending:End, Y = ending1:End, NN is N +1.

%p/vcn
check_dh([N,_,_,p,X],[NN,_,_,vcn,Y]):- !, NN is N +1.

%vcn/n
check_dh([N,_,_,vcn,_],[NN,_,_,X,_]):- !,  (X = n; X = pro), NN > N.

/******************TP************************************************/
% verb/tp
check_dh([N,_,_,v,X],[NN,_,_,tp,Y]) :- !,  X = ending:End, Y = ending:End, NN is N + 1.
check_dh([N,_,_,v,_],[NN,_,_,tp,_]) :- !, NN is N + 1.
```

```
/**************P***********************************************************/
% p(verbal ending)/postpositional marker(pp)
%check_dh([N,_,_,p,X],[NN,_,_,vcn,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.

% verb/p
check_dh([N,_,_,v,_],[NN,_,_,p,_]) :- !, NN is N+1.

% tp/p
check_dh([N,_,_,tp,_],[NN,_,_,p,_]) :- !, NN is N +1.

% n/p  Noun ending in vow only
check_dh([N,_,_,n,X],[NN,_,_,p,_]) :- !, X = ending:vow, NN is N + 1.

/***************SERV*******************************************************/
check_dh([N,_,_,v,_],[NN,_,_,serv,_]) :- !, NN is N + 1.
check_dh([N,_,_,serv,_],[NN,_,_,v,_]) :- !, NN is N + 1.

/**************CP**********************************************************/
% n/cp(copular postmarkers)
check_dh([N,_,_,n,_],[NN,_,_,cp,_]) :-  !, NN is N +1.

% adj/cp
check_dh([N,_,_,adj,_],[NN,_,_,cp,_]) :-  !, NN is N +1.

% cp/tp
check_dh([N,_,_,cp,_],[NN,_,_,tp,_]) :-  !, NN is N +1.

%cp/p
check_dh([N,_,_,cp,_],[NN,_,_,p,_]) :-  !, NN is N +1.

% cp/vcn
check_dh([N,_,_,cp,_],[NN,_,_,vcn,_]) :-  !, NN is N +1.

%pp/cp
check_dh([N,_,_,pp,_],[NN,[[M,_,_,n,_]],_,cp,_]) :-  !, N < NN, NN is M + 1.

/****************CCP ****************************************************/
% n/ccp
check_dh([N,_,_,n,_],[NN,_,_,ccp,_]) :-  !, NN is N +1.

% advp/ccp
check_dh([N,_,_,advp,X],[NN,_,_,ccp,_]) :- !, N < NN.

% pp/ccp
check_dh([N,_,_,pp,X],[NN,_,_,ccp,_]) :- !,  N < NN.

% one pp/ccp
check_dh([N,_,_,pp,X],[NN,[],_,ccp,_]) :- !, X = case:nom, N < NN.

% two pp/ccp
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Y]],_,ccp,_]) :- !, \+ (X = case:acc, Y = case:acc), N < NN, M < NN.

% pp & advp/ccp
check_dh([N,_,_,advp,X],[NN,[[M,_,_,pp,Y]],_,ccp,_]) :- !, M is N + 1, N < NN.
```

```
/******************v:sta*****************************************************/
% advp/v
check_dh([N,_,_,advp,X],[NN,[],_,v,Y]) :- !,(Y = feat:sta; Y=feat:dyn),  N < NN .

% one pp/v
check_dh([N,_,_,pp,X],[NN,[],_,v,Y]) :- !,  (((Y = feat:sta;Y=feat:dyn; Y =subcat:2), (X = case:nom; X
=case:nom-also)); (X= case:acc, Y = subcat:2)), N < NN.

% two pp/v
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Y]],_,v,Z]) :- !,\+ (Z = feat:sta, (X = case:acc ; Y = case:acc)),
                                \+ (Z = feat:dyn, (X = case:nom ; X=case:nom-topicalization; X=case:nom-
also),
                                                (Y = case:nom; Y=case:nom-topicalization; Y=case:nom-
also)), NN > M, NN > N.

% two pp&ncv /v
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Y],[O,_,_,ncv,_]],_,v,Z]) :- !,\+ (Z = feat:sta, (X = case:acc ; Y =
case:acc)),
                                \+ (Z = feat:dyn, (X = case:nom ; X=case:nom-topicalization; X=case:nom-
also),
                                                (Y = case:nom; Y=case:nom-topicalization; Y=case:nom-
also)),
                                                        N < NN, M < NN ,O < NN.

% pp & advp/v
check_dh([N,_,_,advp,X],[NN,[[M,_,_,pp,Y]],_,v,Z]) :- !, (((Y = feat:sta;Y=feat:dyn; Y =subcat:2), X =
case:nom);
                                                (X= case:acc, Y = subcat:2)), M <NN, N < NN.

% n (without case marker) / v
check_dh([N,_,_,n,_],[NN,[],_,v,Y]) :- !,(Y = subcat:2),   NN is N + 1.

check_dh([N,_,_,pp,Z],[NN,[[M,_,_,n,_]],_,v,Y]) :- !,(Y = subcat:2), Z = case:nom, N < NN , M < NN.

/******************GEN*****************************************************/
% n/gen
check_dh([N,_,_,n,X],[NN,_,_,gen,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,_],[NN,_,_,gen,_]) :-  !, NN is N +1.

%gen/n
check_dh([N,_,_,gen,X],[NN,_,_,n,Y]) :-  !, NN > N.

/******************ADVP****************************************************/
% n/advp(adverbial post marker)
check_dh([N,_,baldal,n,_],[NN,_,_,advp,Y]) :- !, Y = ending:vow, NN is N+1.
check_dh([N,_,_,n,X],[NN,_,_,advp,Y]) :-  !, X = ending:End, Y = ending:End, NN is N +1.
check_dh([N,_,_,n,X],[NN,_,_,advp,Y]) :-  !, NN is N +1.

% advp/v
check_dh([N,_,_,advp,X],[NN,_,_,v,Y]) :-  !, NN > N.

% advp/p
check_dh([N,_,_,advp,X],[NN,_,_,p,Y]) :-  !, NN is N +1.

%advp/pp
check_dh([N,_,_,advp,X],[NN,_,_,pp,Y]) :-  !, NN is N +1.
```

```
/****************verb subcat :2*********************************************/
%case marker/verb subcat:2 that has both subject, object, and adverb.
check_dh([N,_,_,pp,X],[NN,[[M,_,_,ZZ,_],[L,_,_,pp,Z]],_,v,Y]) :- !,  Y = subcat:2, (ZZ = adv; ZZ = advp; ZZ =
ncv; ZZ = vcv),
                                        \+ (X = case:Case,Z = case:Case),
                                        N < NN, M < NN, L < NN.

check_dh([N,_,_,ZZ,_],[NN,[[M,_,_,pp,X],[L,_,_,pp,Z]],_,v,Y]) :- !,  Y = subcat:2, (ZZ = adv; ZZ = advp; ZZ =
ncv; ZZ = vcv),
                                        \+ (X = case:Case,Z = case:Case),
                                        N < NN, M < NN, L < NN.

%verb subcat:2 that has subject,object,adv,advp
check_dh([N,_,_,pp,X],[NN,[[L,_,_,adv,_],[O,_,_,advp,_],[M,_,_,pp,Z]],_,v,Y]) :- !,  Y = subcat:2,
                                        \+ (X = case:Case,Z = case:Case),
                                       N < NN, M < NN, L < NN, O < NN.


%verb subcat:2 / nom&acc&vcv&adv
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Z],[L,_,_,adv,_],[O,_,_,ZZ,_]],_,v,Y]) :- !,  Y = subcat:2,
                                        \+ (X = case:Case,Z = case:Case),(ZZ = vcv; ZZ = ncv),
                                       N < NN, M < NN, L < NN, O < NN.

%case marker/verb subcat:2(nom & complementizer)
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pcv,_]],_,v,_]) :- !, X = case:nom, N < NN, M < NN.

/********verb subcat:1*********************************************************/
% 'because of verb'
check_dh([N,_,_,n,X],[NN,[],ddae-muni,v,_]) :- !,X = ending:vow, NN is N + 1.
check_dh([N,_,i,pp,_],[NN,[],ddae-muni,v,_]) :- !, NN is N + 1.

check_dh([N,_,_,pp,X],[NN,[[M,_,_,Z,_]],ddae-muni,v,_]) :- !,X = case:nom-topicalization, (Z = n ; Z = pp),
                                        NN is M + 1, NN > N.


%case marker/verb subcat:1 OR subcat:2 that has one dep. realized.
check_dh([N,_,_,pp,X],[NN,[],_,v,Y]) :- !,   (Y = subcat:2, X = case:acc );
                                        (Y = subcat:1 ,
                                        (X = case:nom ;X = case: nom-topicalization)),

                                    N < NN.

% two nom marker / verb subcat:1 & feat:sta
check_dh([N,_,_,pp,X],[NN,[[M,_,_,pp,Z]],_,v,Y]) :- !, (Y = sucat:1 , Y = feat:sta),
                                        (X = case:nom ;X = case: nom-topicalization;X = case:nom-also),
                                                (Z = case:nom ;Z = case: nom-topicalization;Z = case:nom-
also),
                                                N < NN, M < NN.

% verb subcat:1  / nom & adv or advp
check_dh([N,_,_,pp,X],[NN,[[M,_,_,adv,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                        (X = case:nom ;X = case: nom-topicalization;X = case:nom-also));
                                        (Y = subcat:2, X = case:acc)),
                                       N < NN, M < NN.
```

```
% verb subcat:1  / nom & adv or advp
check_dh([N,_,_,pp,X],[NN,[[M,_,_,advp,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization;X = case:nom-also));
                                    (Y = subcat:2, X = case:acc)),
                                   N < NN, M < NN.

% verb subcat:1  / nom & ncv
check_dh([N,_,_,pp,X],[NN,[[M,_,_,ncv,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization;X = case:nom-also));
                                    (Y = subcat:2, X = case:acc)),
                                   N < NN, M < NN.

% verb subcat:1 / nom & advp & ncv
check_dh([N,_,_,pp,X],[NN,[[M,_,_,advp,_],[O,_,_,ncv,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization));
                                    (Y = subcat:2, X = case:acc)),
                                N < NN, M < NN, O < NN.

%verb subcat:1 / nom& adv& advp
check_dh([N,_,_,pp,X],[NN,[[M,_,_,adv,_],[L,_,_,advp,_]],_,v,Y]) :- !,  (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization));
                                    (Y = subcat:2, X = case:acc)),
                                     N < NN, M < NN, L < NN.

%verb subcat:1 /pcv & nom
check_dh([N,_,_,pp,_],[NN,[[M,_,_,pcv,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization));
                                    (Y = subcat:2, X = case:acc)),
                                N < NN, M < NN.

%verb subcat:1 /nom & vcv
check_dh([N,_,_,pp,X],[NN,[[M,_,_,vcv,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization));
                                    (Y = subcat:2, X = case:acc)),
                                     N < NN, M < NN.

%verb subcat:1 / nom % vcv &adv
check_dh([N,_,_,pp,X],[NN,[[M,_,_,adv,_],[L,_,_,vcv,_]],_,v,Y]) :- !, (((Y = subcat:1 ; Y = subcat:2),
                                    (X = case:nom ;X = case: nom-topicalization));
                                    (Y = subcat:2, X = case:acc)),
                                            N < NN, M < NN, L < NN.

%case marker/verb subcat:2 that has either subject or object, and an adverb/advp/ncv.
check_dh([N,_,_,adv,_],[NN,[[M,_,_,pp,O]],_,v,Y]) :- !, (Y = subcat:2),O = case:acc,
                                    N < NN, M < NN.
check_dh([N,_,_,advp,_],[NN,[[M,_,_,pp,O]],_,v,Y]) :- !, (Y = subcat:2),O = case:acc,
                                        N < NN, M < NN.
check_dh([N,_,_,ncv,_],[NN,[[M,_,_,pp,O]],_,v,Y]) :- !, (Y = subcat:2),O = case:acc,
                                        N < NN, M < NN.
check_dh([N,_,_,vnp,_],[NN,[[M,_,_,pp,O]],_,v,Y]) :- !, (Y = subcat:2),O = case:acc,
                                        N < NN, M < NN.
/*************************************************************************************/
/*************ADV*********************************************************************/
%adv/verb
check_dh([N,_,_,adv,_],[NN,[],_,v,_]) :- !, NN > N.
```

```
%adv/verb.
check_dh([N,_,_,adv,_],[NN,X,_,v,_]) :- !, N < NN, \+ (member([M,_,_,adv,_],X)).



%adv/adv
check_dh([N,_,_,adv,X],[NN,_,_,adv,Y]) :-  !, NN is N +1.

%adv/adj
check_dh([N,_,_,adv,X],[NN,_,_,adj,Y]) :-  !, NN is N +1.

/***************ADJ*************************************************************/
% adjective/noun
check_dh([N,_,_,adj,_],[NN,_,_,n,_]) :- !, NN is N+1.

/*********************VNP*********************************************************/
% v/vnp(post marker that makes verb into noun)
check_dh([N,_,_,v,X],[NN,_,_,vnp,Y]):- !, X = ending:End, Y = ending1:End, NN is N+1.
check_dh([N,_,_,v,X],[NN,_,_,vnp,Y]):- !, NN is N+1.

% tp/vnp
check_dh([N,_,_,tp,X],[NN,_,_,vnp,Y]):- !, NN is N+1.

%vnp/pp
check_dh([N,_,_,vnp,X],[NN,_,_,pp,Y]):- !, X = ending2:End , Y = ending:End, NN is N+1.

%vnp/n
check_dh([N,_,_,vnp,X],[NN,_,_,n,Y]):- !, NN is N+1.

/*********************NNP*********************************************************/
%n/ nnp
check_dh([N,_,_,n,_],[NN,_,_,nnp,_]):- !,  NN is N+1.

% nnp/v(ha)
check_dh([N,_,_,nnp,_],[NN,_,ha,v,_]):- !,  NN is N+1.

%nnp/cp(i)
check_dh([N,_,_,nnp,_],[NN,_,_,cp,_]):- !,  NN is N+1.



%%%%%parsing algorithm itself%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


parse(InputList,Result) :-
  parse_loop(1,InputList,[],Result).


% parse_loop(+Count,+InputList,+WordList,+HeadList,-Result)
%  Called by parse/2, to loop through the list.


parse_loop(Count,[Word|InputList],HeadList,Result) :-
  look_for_word(Count,Word,Node),
  parse_node(Node,HeadList,NewHeadList), % Try to attach it
  NewCount is Count + 1,
```

```
   parse_loop(NewCount,InputList,NewHeadList,Result).


%%%%To allow words that are not in the lexicon to be considered as a noun or verb.

look_for_word(Count,Word,Node) :-  word(Word,WordFeatures),
                        WordFeatures =.. X,
                                Node= [Count,[],Word|X].

look_for_word(Count,Word,Node) :- \+ word(Word,_), Count < 15, Node = [Count,[],Word,n|_].

look_for_word(Count,Word,Node) :- \+ word(Word,_), Count > 15, Node = [Count,[],Word,v|_].


% No more words to parse; so Result := HeadList.
% The pp,p,and v cannot come alone in a sentence.
parse_loop(_,[],[H],[H]).


parse_node(Node,[],[Node]).
   % No more elements of HeadList to look at.


parse_node(Node,[Head|HeadList],NewHeadList) :-
   % Insert Node above Head, and remove Head from HeadList.
   Node = [N,[],W|X],
   Head = [NN,_,WW|Y],
   check_dh(Head,Node),      % Head is the dependent here

   NewNode = [N,[Head],W|X], %Everytime the head%dependent relation has been checked, the Node
changes.
   parse_node(NewNode,HeadList,NewHeadList).


parse_node(Node,[Head|HeadList],NewHeadList) :-
   % Insert Node above Head, and remove Head from HeadList.
   Node = [N,[D1],W|X],
   Head = [NN,_,WW|Y],
   check_dh(Head,Node),      % Head is the dependent here
   append([D1],[Head],NewD),
   NewNode = [N,NewD,W|X], %Everytime the head%dependent relation has been checked, the Node
changes.

   parse_node(NewNode,HeadList,NewHeadList).


parse_node(Node,[Head|HeadList],NewHeadList) :-
   % Insert Node above Head, and remove Head from HeadList.
   Node = [N,[D1,D2],W|X],
   Head = [NN,_,WW|Y],
   (check_dh(Head,Node);
    check_dh(Head,[N,[D2,D1],W|X])
    ),% Head is the dependent here
   append([D1,D2],[Head],NewD),
   NewNode = [N,NewD,W|X], %Everytime the head%dependent relation has been checked, the Node
changes.
```

```
   parse_node(NewNode,HeadList,NewHeadList).

parse_node(Node,[Head|HeadList],NewHeadList) :-
   % Insert Node above Head, and remove Head from HeadList.
   Node = [N,[D1,D2,D3],W|X],
   Head = [NN,_,WW|Y],
   (check_dh(Head,Node);
    check_dh(Head,[N,[D2,D1,D3],W|X]);
    check_dh(Head,[N,[D2,D3,D1],W|X]);
    check_dh(Head,[N,[D1,D3,D2],W|X]);
    check_dh(Head,[N,[D3,D1,D2],W|X]);
    check_dh(Head,[N,[D3,D2,D1],W|X])
    ),% Head is the dependent here
   append([D1,D2,D3],[Head],NewD),
   NewNode = [N,NewD,W|X], %Everytime the head%dependent relation has been checked, the Node
changes.

    parse_node(NewNode,HeadList,NewHeadList).

%%"projectivity constraint"  ;not allowing projectivity
%when there is no link, add to the front of HeadList
parse_node(Node,HeadList,NewHeadList):- NewHeadList = [Node|HeadList].



%"projectivity"   ;allowing projectivity
%when there is no link, add to the front of HeadList.
%parse_node(Node,[Head|HeadList],[Head|NewHeadList]) :-
%     parse_node(Node,HeadList,NewHeadList).


% Output utilities


write_list([First|Rest]) :-
   var(First),
   !,
   write('_ '),
   write_list(Rest).

write_list([First|Rest]) :-
   write(First),
   write(' '),
   write_list(Rest).

write_list([]) :-
   nl.


write_dep(Node) :-
   write_dep2(Node,0).

write_dep2([N,Dependents|Rest],Indentation) :-
   tab(Indentation),
   write(N),
```

```
  write(' '),
  NewIndentation is Indentation + 3,tab(Indentation),
  display_feature_structure(Rest),   %% displays the GULP features
  write_dep3(Dependents,NewIndentation).

write_dep3([First|Rest],N) :- write_dep2(First,N), write_dep3(Rest,N).

write_dep3([],_):- !.

% tests

try(List) :- parse(List,[Head]),
          write_dep(Head).
           %  nl,
           % fail.

try(_) :- write('No (more) parses.'), nl.
```

B. SEVERAL EXAMPLES OF INPUT AND OUTPUT OF TEST SENTENCES

```
?- try([ulma,jen,ggaji,nun,son,euro,ilha,-nun,sawhe,-i,ss-b-ni,da]).
12 [da, p, sem:declarative]
  11    [ss-b-ni, tp, ending:vow..tense:past-honorific]
    10      [-i, cp, sem:be]
      9         [sawhe, n, ending:vow..sem:society]
        8           [-nun, vcn, ending:vow..tense:pres..ending1:vow]
          7             [ilha, v, ending:vow..sem:work..subcat:1..feat:dyn]
            6               [euro, advp, ending:con..sem:to-or-as-or-with]
              5                 [son, n, ending:con..sem:hand]
            4               [nun, pp, ending:vow..case:nom-topicalization..num:sing]
              3                 [ggaji, advp, ending:con..sem:uptil]
                2                   [jen, n, ending:con..sem:before-or-past-entire]
                  1                     [ulma, adj, sem:some-amount]

Yes

?- try([intenet,gisul,euy,baldal,ro,sege,nun,jemjem,de,gaggawhe-ji,-go,jengbo,yang,do,emchung-
nage,nul,ess,da]).
18 [da, p, sem:declarative]
  17    [ess, tp, ending:con..tense:past]
    16      [nul, v, ending:con..sem:increase..subcat:1..feat:dyn]
      15        [emchung-nage, adv, sem:enormously]
      14        [do, pp, ending:con..case:nom-also..num:sing]
        13          [yang, n, ending:con..sem:quantity]
          12            [jengbo, n, ending:vow..sem:information]
      11        [-go, vcv, ending:vow..sem:also-or-as-or-for]
        10          [gaggawhe-ji, v, ending:vow..sem:become-close..subcat:1..feat:dyn]
          9             [de, adv, sem:more]
            8               [jemjem, adv, sem:gradually]
          7             [nun, pp, ending:vow..case:nom-topicalization..num:sing]
            6               [sege, n, ending:vow..sem:world]
          5             [ro, advp, ending:vow..sem:to-or-as]
            4               [baldal, n, ending:con..sem:growth-or-progress-or-advancement]
              3                 [euy, gen, ending:con..sem:of]
```

```
          2                [gisul, n, ending:con..sem:skill]
            1                  [intenet, n, ending:con..sem:internet]

Yes

?-  try([tom,i,susan,ul,john,i,po,ss,da,go,malha,n,da]).


13 [da, p, sem:declarative]
  12   [n, tp, ending:vow..tense:pres]
    11     [malha, v, ending:vow..sem:say..subcat:2..feat:dyn]
      10       [go, pcv, sem:that]
        9         [da, p, sem:declarative]
          8           [ss, tp, ending:vow..tense:past]
            7             [po, v, ending:vow..sem:see..subcat:2..feat:dyn]
              6               [i, pp, ending:con..case:nom..num:sing]
                5                 [john, n, ending:con..sem:john]
              4               [ul, pp, ending:con..case:acc..num:sing]
                3                 [susan, n, ending:con..sem:susan]
      2       [i, pp, ending:con..case:nom..num:sing]
        1         [tom, n, ending:con..sem:tom]

Yes

?- try([teuk-hi,nam-nye,euy,pyung-kyun, kyung-umwha,senho-do,nun,yeuja,ga,dwen-sori,bal-
eum,ul,de,senhoha,-nun,geut,euro,deule-na,ss,da]).
20 [da, p, sem:declarative]
  19   [ss, tp, ending:vow..tense:past]
    18     [deule-na, v, ending:vow..sem:be-revealed..subcat:1..feat:dyn]
      17       [euro, advp, ending:con..sem:to-or-as-or-with]
        16         [geut, n, ending:con..sem:thing-or-fact]
          15           [-nun, vcn, ending:vow..tense:pres..ending1:vow]
            14             [senhoha, v, ending:vow..sem:prefer..subcat:2..feat:dyn]
              13               [de, adv, sem:more]
              12               [ul, pp, ending:con..case:acc..num:sing]
                11                 [bal-eum, n, ending:con..sem:pronunciation]
                  10                   [dwen-sori, n, ending:vow..sem:strong-sounds]
              9              [ga, pp, ending:vow..case:nom..num:sing]
                8                [yeuja, n, ending:vow..sem:woman]
      7       [nun, pp, ending:vow..case:nom-topicalization..num:sing]
        6         [senho-do, n, ending:vow..sem:preference]
          5           [kyung-umwha, n, ending:vow..sem:change-to-strong-sounds]
            4             [pyung-kyun, n, ending:con..sem:average]
              3               [euy, gen, ending:vow..sem:of]
                2                 [nam-nye, n, ending:vow..sem:boys-and-girls]
      1       [teuk-hi, adv, sem:especially]

Yes


?- try([gang-a-gi,nun,ho-rang-i,ga,chot,-nun,geut,ul,al,ess,da]).
11 [da, p, sem:declarative]
  10   [ess, tp, ending:con..tense:past]
```

```
    9      [al, v, ending:con..sem:know..subcat:2..feat:dyn]
      8        [ul, pp, ending:con..case:acc..num:sing]
        7          [geut, n, ending:con..sem:thing-or-fact]
          6            [-nun, vcn, ending:vow..tense:pres..ending1:con]
            5              [chot, v, ending:con..sem:chase..subcat:2..feat:dyn]
              4                [ga, pp, ending:vow..case:nom..num:sing]
                3                  [ho-rang-i, n, ending:vow]
      2        [nun, pp, ending:vow..case:nom-topicalization..num:sing]
        1          [gang-a-gi, n, ending:vow..sem:dog]

Yes

?- try([21,segi,ey, jen,sege,saerob,-un,jishik,ul,gajang,manni, changjoha,-nun,nara,ga,sege,lul, ii-ggule-ga,-
l,geut,i-la-go,yae-chuk-ha,b-ni,da]).
24 [da, p, sem:declarative]
  23   [b-ni, tp, ending:vow..tense:pres-honorific]
    22     [yae-chuk-ha, v, ending:vow..sem:predict..subcat:2..feat:dyn]
      21       [i-la-go, ncv, ending:con..sem:says-that..subcat:1]
        20         [geut, n, ending:con..sem:thing-or-fact]
          19           [-l, vcn, ending:vow..tense:future..ending1:vow]
            18             [ii-ggule-ga, v, ending:vow..sem:lead..subcat:2..feat:dyn]
              17               [lul, pp, ending:vow..case:acc..num:sing]
                16                 [sege, n, ending:vow..sem:world]
              15               [ga, pp, ending:vow..case:nom..num:sing]
                14                 [nara, n, ending:vow..sem:country]
                  13                   [-nun, vcn, ending:vow..tense:pres..ending1:vow]
                    12                   [changjoha, v, ending:vow..sem:create..subcat:2..feat:dyn]
                      11                   [manni, adv, sem:a-lot]
                        10                   [gajang, adv, sem:the-most]
                      9                  [ul, pp, ending:con..case:acc..num:sing]
                        8                    [jishik, n, ending:con..sem:knowledge-or-information]
                          7                    [-un, vcn, ending:con..tense:past..ending1:con]
                            6                    [saerob, v, ending:con..sem:be-new..subcat:1..feat:sta]
                          5                    [sege, n, ending:vow..sem:world]
                            4                    [jen, n, ending:con..sem:before-or-past-entire]
                      3                  [ey, advp, ending:vow..sem:of-or-to-or-on-or-at-or-in-or-in-addition-
to]
                        2                    [segi, n, ending:vow..sem:century]
                          1                    [21, num, sem:twenty-one]

Yes

?- try([21,segi,euy,ju-yek,i,dwe,-l,uli,e-linhi,dul,i,ii,jeun-mang,ul,gaseum,ey,jal, saegye-du,-l,pil-yo,ga,it,seb-
ni,da]).
24 [da, p, sem:declarative]
  23   [seb-ni, tp, ending:con..tense:pres-honorific]
    22     [it, v, ending:con..sem:be-or-exist..subcat:1..feat:dyn]
      21       [ga, pp, ending:vow..case:nom..num:sing]
        20         [pil-yo, n, ending:vow..sem:necessity]
          19           [-l, vcn, ending:vow..tense:future..ending1:vow]
            18             [saegye-du, v, ending:vow..sem:keep-in-mind..subcat:2..feat:dyn]
              17               [jal, adv, sem:well]
              16               [ey, advp, ending:con..sem:of-or-to-or-on-or-at-or-in-or-in-addition-to]
                15                 [gaseum, n, ending:con..sem:heart]
              14               [ul, pp, ending:con..case:acc..num:sing]
                13                 [jeun-mang, n, ending:con..sem:foresight-or-perspective]
```

12 [ii, pro, ending:vow..sem:this-or-it]
11 [i, pp, ending:con..case:nom..num:sing]
10 [dul, plp, ending:con..num:pl]
9 [e-linhi, n, ending:vow..sem:child]
8 [uli, n, ending:vow..sem:us]
7 [-l, vcn, ending:vow..tense:future]
6 [dwe, ccp, sem:become]
5 [i, pp, ending:con..case:nom..num:sing]
4 [ju-yek, n, ending:con..sem:leading-part]
3 [euy, gen, ending:vow..sem:of]
2 [segi, n, ending:vow..sem:century]
1 [21, num, sem:twenty-one]

Yes

%%%% The Appendices will be modified by probably abbreviating the code of KorPar and listing examples of test sentences that were mentioned in the previous chapters. %%%%%%