Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Che Yeol (Jayeol) Chun April 10, 2018

Dependency Analysis of Abstract Universal Structures in Korean and English

by

Che Yeol (Jayeol) Chun

Jinho D. Choi, Ph.D.
Adviser

Department of Mathematics and Computer Science

Jinho D. Choi, Ph.D.

Adviser

Jeremy A. Jacobson, Ph.D.

Committee Member

Phillip Wolff, Ph.D.

Committee Member

2018

Dependency Analysis of Abstract Universal Structures in Korean and English

By

Che Yeol (Jayeol) Chun

Jinho D. Choi, Ph.D.

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics and Computer Science

2018

Abstract

Dependency Analysis of Abstract Universal Structures in Korean and English
By Che Yeol (Jayeol) Chun

This thesis gives two contributions in the form of lexical resources to (1) dependency parsing in Korean and (2) semantic parsing in English. First, we describe our methodology for building three dependency treebanks in Korean derived from existing treebanks and pseudo-annotated according to the latest guidelines from the Universal Dependencies (UD). The original Google Korean UD Treebank is re-tokenized to ensure morpheme-level annotation consistency with other corpora while maintaining linguistic validity of the revised tokens. Phrase structure trees in the Penn Korean Treebank and the Kaist Treebank are automatically converted into UD dependency trees by applying head-percolation rules and linguistically motivated heuristics. A total of 38K+ dependency trees are generated. To the best of our knowledge, this is the first time that the three Korean treebanks are converted into UD dependency treebanks following the latest annotation guidelines. Second, we introduce an on-going project for constructing a new corpus of Deep Dependency Graphs (DDG) which are converted from the phrase structure trees in the OntoNotes corpus with additional semantic information found in the Proposition Bank (PropBank) and Abstract Meaning Representation (AMR). This new dataset plays a pivotal role in our proposed novel AMR parsing scheme in which the data helps train a dependency parser, which is subsequently trained on a new AMR parsing task through transfer learning. Since AMR inherits the core semantic roles in PropBank, we speculate that the first training phase that exposes the parsing model to semantic role labeling task will greatly help the model perform AMR parsing. In this thesis, we address the preliminary step of integrating PropBank labels for predicate argument relations during the constituent-to-dependency conversion of the OntoNotes. It is our hope that the new corpus, with its rich syntactic information stored in DDG as well as semantic role information provided by PropBank that fully describes the predicate argument structure, will serve as a useful resource for semantic role labeling.

Dependency Analysis of Abstract Universal Structures in Korean and English

By

Che Yeol (Jayeol) Chun

Jinho D. Choi, Ph.D.

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics and Computer Science

2018

Acknowledgements

I would like to send my deepest appreciation to my parents in Korea, Sang Kyu Chun and Ji Eun Lee, who have supported me during my 8 years at Emory University. In their unconditional support I found an endless source of strength and inspiration, and I remain forever grateful for the opportunity to immerse myself with all the valuable resources Emory had to offer. I am also thankful to my younger brother, Jae Woo Chun, who grew up to be the dependable younger brother while I was away.

I also want to acknowledge my academic advisor Dr. Frank Gaertner, whom I was fortunate to have met as part of a first generation of military returnees he mentored in 2014. His presence served as a constant reminder of the liberal arts spirit, and without his guidance I would not have enjoyed such a colorful undergraduate career at Emory University.

I am also greatly indebted to my thesis advisor Dr. Jinho Choi, who introduced me to the Natural Language Processing and gave me an opportunity to conduct meaningful research in the field. Under his mentorship I had the privilege of seeing first-hand the qualities required to be a researcher, and I inherit his passion for the subject. I am grateful to his guidance that made this thesis possible.

My final acknowledgments go out to Dr. Valerie Summet, now at Rollins College, who encouraged me to major in Computer Science; Dr. Daniel Weissman, my Physics advisor, who helped initiate my first research project; Dr. Jeremy Jacobson, under whose leadership I was lucky to have served the Emory Data Science Club; and Dr. Phillip Wolff, who took interest in my research and agreed to be on my honors committee.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recent advances in Natural Language Processing (NLP) have been tremendous. Facilitated by the advent of deep learning and computing power that supports heavy computation, research to automatically process and understand language has become the backbone of such systems as chat-bots and machine translations, many of which are already commercially available.

One research area that benefits from the increased computing capacity to aid natural language understanding is semantic parsing[1], which discovers meaning representations within a sentence. It is therefore not surprising that many NLP applications are preceded by parsing in order to understand the meaning of the sentence[2]. Since the output of the parser, known as a parse tree or a parse graph, can become an input to other NLP applications like question answering, development of an accurate parser can have far-reaching benefits.

In order to build an effective parsing model, one needs (1) a sufficiently large training dataset where each source sentence is mapped to a representation of its meaning, known as meaning representation, and (2) an optimized model to process training algorithm in a realistic duration of time while learning the meaningful patterns of the natural language

[1]A closely related term is syntactic parsing, which discovers syntactic structures within a sentence.

[2]In this respect, parsing may be regarded as a machine equivalent of human language comprehension.

it encounters during training. While a generic learning algorithm continues to be refined and improved on efficiency and accuracy of solving tasks in and outside NLP, it remains to be seen how the availability of large annotated corpora and other lexical resources may help develop an advanced representation of meanings that leads to a more accurate parsing model.

Dependency Parsing in Korean Korean is an agglutinative language with a flexible word order such that, although Korean generally follows the SOV (subject-object-verb) sentence construction, any other orders are still acceptable with further possibilities for omissions of any of the above constituents. For this reason, it has been suggested that the dependency structure which is not inhibited by word-order may be more suitable for capturing the semantics of languages like Korean [6]. Furthermore, the inspection of morphemes (Section 2.1.2) can easily lead to inference of dependency relations, which can help boost the parsing performance.

Despite these appeals, however, dependency parsing has not received its due attention in the Korean NLP community. This is largely due to the lack of large-scale dependency corpora. While several treebanks have been introduced for Korean [22, 12, 13], they all contain annotation of phrase structure trees. Previous dependency parsing experiments therefore had to be preceded by constituent-to-dependency conversion using head-percolation rules and linguistically motivated heuristics [6, 3] (See Section 2.2 for details on the conversion methodology). The parsing results were promising [6]: yet, the previous efforts produced a distinct set of dependency labels for each corpus, introducing incompatibility among the resultant dependency trees. Thus, in order to create a single large corpus of compatible dependency trees that is conducive to parsing experiments and subsequent comparative analysis, it is necessary to convert phrase structure trees from different treebanks into dependency trees with a consistent set of relations.

Semantic Parsing in English In the other research direction, growing interest in semantic parsing has culminated in the creation of Abstract Meaning Representation (AMR). AMR [2] is a semantic representation language that expresses what had traditionally existed as separate annotations, such as named entities and semantic relations, in a single annotation as a directed acyclic graph. AMR inherits the core semantic role labels from the Propposition Bank (PropBank) [19] and uses them as the label of the relation between vertices of the graph. Furthermore, Deep Dependency Graph (DDG) [4] is a dependency structure captures the complete predicate argument structure at the cost of losing some of the tree properties defined in Section 2.1.4.2.

These motivate a direct insertion of the PropBank labels into DDG converted from phrase structure trees in the OntoNotes corpus [25], where the dependency labels between a predicate and its arguments are completely replaced by the corresponding PropBank labels. This produces a new corpus, which we title PropBank-Augmented OntoNotes DDG corpus, that retains the DDG dependency structure and its rich syntactic features with additional semantic role information provided by PropBank. It is our expectation that a dependency parser trained on this new corpus will have learned the semantics of the predicate argument structure, which is required for semantic role labeling. Then, transfer learning can be applied by training the dependency parser on the AMR parsing task. Since AMR inherits the semantic role labels from PropBank and displays resemblance to the dependency structure, we conjecture that the initial training phase which exposes the model to semantic role labeling and dependency structure will help the parser perform the AMR parsing task. To this end, we believe that the new corpus will be an ideal resource for sematic role labeling and, ultimately, AMR parsing.

1.2 Objectives

The objectives of this thesis are as follows:

- Re-tokenization of Google UD Korean Treebank and systematic assessment for errors present in the original corpus
- Constituent-to-dependency conversion of the phrase structure trees in the Penn Korean Treebank and the Kaist Treebank
- Analysis of the three converted Korean dependency treebanks and remaining issues

- Construction of new corpus by replacing dependencies that represent predicate argument structure in DDG converted from the OntoNotes corpus with PropBank labels

- Analysis of the new corpus in creation and remaining challenges

Chapter 2

Background

2.1 Natural Language Structures

2.1.1 Parts of Speech

One of the simplest methods of extracting information from each word in a sentence (also known as a token) is by looking at its Parts-of-Speech (POS). POS refers to a category of lexical units that share similar grammatical properties and communicates the syntactic function of the token. We describe 5 common POS below.

- Adjective (`ADJ`): describes properties or attributes of noun.
- Punctuation (`P`): functions as a delimiter to facilitate understanding.
- Noun (`NOUN`): denotes a real or abstract thing or a group of it, such as people, objects, and ideas.
- Verb (`VERB`): describes the actions performed at or by the subject.
- Adverb (`ADV`): modifies other POS such as adjectives, adverbs, nouns and verbs and supplies information such as manner and time.

Additionally, linguistically meaningful combinations of tokens may be grouped into a phrase, which receives a phrasal POS (Figure 2.1).

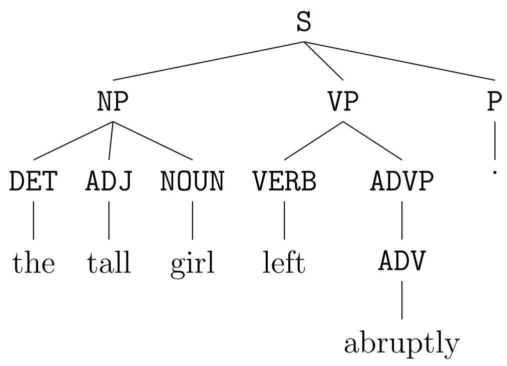


Figure 2.1: Example phrase structure tree annotation of a sentence "*The tall girl left abruptly.*"
`S`: Sentence, `NP`: Noun Phrase, `VP`: Verb Phrase, `ADVP`: Adverbial Phrase

2.1.2 Morphological Analysis

Just as tokens are constituents of a sentence, morphemes are constituents of a token. Consider the Korean word 말했다, which can be broken down into 말하/*say*+었/*past*+ 다/*final*, where a plus sign (+) separates each morpheme. 말하/*say* is the verb root that establishes the primary meaning of the token, while 었/*past* denotes the past tense. Finally, 다/*final* marks the end of the sentence. Such analysis of discovering the morphemes that make up a token is known as morphological anlaysis, and it often sheds light on the syntactic function as well as meaning of the token.

Many different languages feature different levels of morphology. Morphological analysis is particularly relevant to a family of languages called morphologically rich languages (MRLs), where dealing solely with surface forms inevitably leads to a data sparsity issue. Unlike English, Korean belongs to this family. Therefore, any parser development for the language must be preceded by either a corpus creation with morpheme-level annotation or morphological analysis of the corpus obtained by an automatic morphological analyzer.

2.1.3 Phrase Structure

Phrase structure grammar, also known as constituency grammar, denotes a way of viewing a sentence as a combination of its constituents. A constituent is a word or a phrase that acts as a single unit and thus can be assigned the same POS. The phrase structure representation of the sentence "*The tall girl left abruptly.*" is shown in Figure 2.1. In the figure, the three constituents *the*, *tall*, and *girl* combine to form a noun phrase, which is a constituent that constitutes a sentence along with a verb phrase and a period.

The Penn Treebank [16] is a phrase structure treebank that has influenced the annotation style of many other treebanks, such as the OntoNotes corpus [25] and the Penn Korean Treebank [12]. Below we describe two of the most prominent characteristics in the Penn Treebank style constituency trees: function tags and empty categories.

Function Tags In its most basic form, such as the one shown in Figure 2.1, the annotated phrase structure tree features nothing more than a constituent structure of the sentence and POS of individual nodes and phrases. However, the annotation can be augmented with an addition of function tags, which supply additional information regarding the semantic role played by a node. There exist various types of functions tags. For example, `-SBJ` denotes a subject of a clause or a sentence, as seen with the leftmost child of `S` in Figure 2.2. Although the function tagset defined for the Penn Korean Treebank differs from the tagset for the OntoNotes corpus—likely as a result of langague-specific extensions—the function tags serve as an extremely helpful guide during the constituent-to-dependency conversion for identifying a dependency relation.

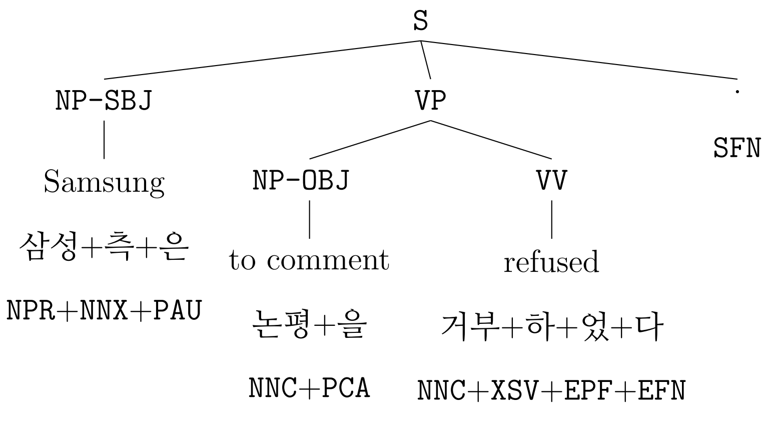


Figure 2.2: Example Penn Korean Treebank style phrase structure tree with function tags of a sentence "*삼성측은 논평을 거부했다.*"
`-SBJ`: subject, `-OBJ`: object

Empty Categories Empty categories are nominal units that indicate the location of their antecedent syntactic elements or dropped elements. In dependency structure, they serve to capture long-distance dependencies at the cost of introducing *non-projectivity* in the resultant tree (See Section 2.1.4 for a discussion on *projectivity*). Although the OntoNotes corpus and the Penn Korean Treebank both feature empty categories, the types of empty categories present in each corpus again differ. Below we list four empty categories defined for the Penn Korean Treebank.

- `Trace`: An argument that precedes its subject leaves in its place a trace `*T*`.
- `Ellipsis`: A predicate that is dropped in a matrix clause or in a clausal coordination of implicitly shared predicate is represented by `*?*`.
- `Empty Assignment`: Dropped arguments are represented by `*pro*`.
- `Empty Operator`: Relative clauses are represented by `*op*`.

Figure 3.7 in Section 3.1.2.1 shows an example phrase structure tree featuring all four types of empty categories.

2.1.4 Dependency Structure

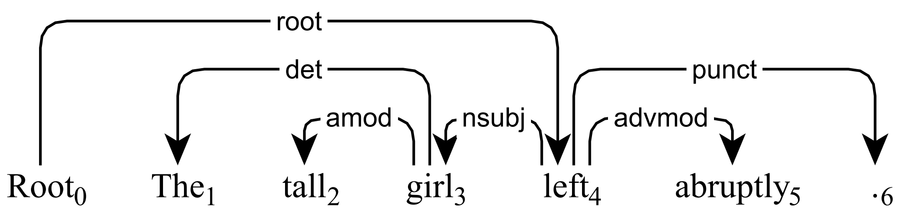


Figure 2.3: Example dependency annotation of a sentence "*The tall girl left abruptly.*" See Table 4.3 for a description of the dependency labels used.

As opposed to the constituency grammar, dependency grammar establishes a directed arc between two nodes in a sentence, where the link is known as a dependency. Because the link is directed, a head-child relationship is recursively established until there remains a single root that can traverse to any other node in the tree. It carries syntactic and semantic information through a dependency label that defines how the nodes are related.

Figure 2.3 and 2.4 show the dependency representation of a sentence "*The tall girl left abruptly.*"

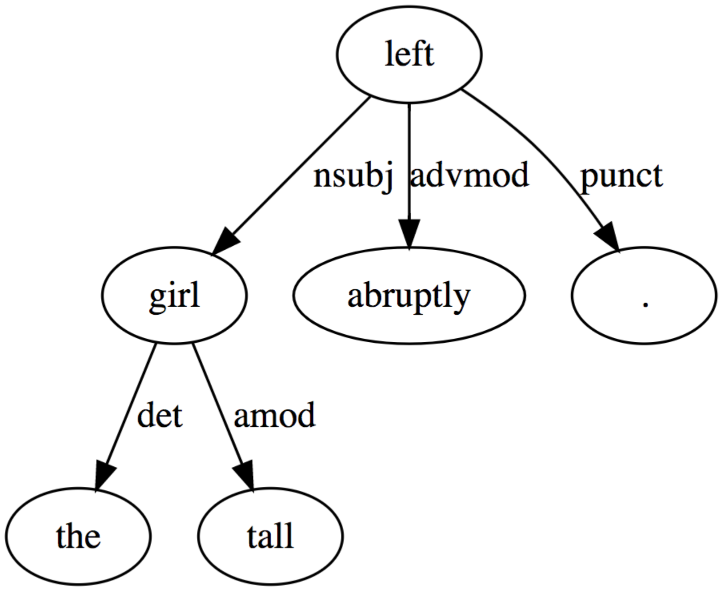


Figure 2.4: Tree visualization of example dependency annotation of a sentence "*The tall girl left abruptly.*"

However, the dependency representation needs not always yield a tree. In fact, a dependency structure is often represented as a directed graph. For all possible graphs represented through dependency structure, dependency trees correspond to a subset of them which satisfy the following properties of *well-formed* dependency graphs.

- There exists a *unique root*, which is not dependent on any other node in the sentence.
- Each vertex is dependent on a *single head*, i.e. each vertex has a single incoming arc.
- The graph is *connected*, i.e. there exists a path to traverse between any two nodes in the graph.
- The graph is *acyclic*, i.e. no vertices introduce a cycle into the graph.

Often included in the list is the fifth property of *projectivity*, which enforces that no arcs may cross each other when the graph is drawn in the manner shown in Figure

2.3, with the artifical root node taking the leftmost position. One of the biggest merits of preserving projectivity is the reduction of parsing complexity to a linear time [18]. But non-projective dependencies are often required to adequately represent long-distance dependencies. To this end, this thesis excludes *projectivity* from the properties that define *well-formed* dependency graphs. Since the four properties above implicitly define a tree structure, a well-formed dependency graph is called a dependency tree.

2.1.4.1 Universal Dependencies

There are various ways of representing dependency structure, such as the Stanford Typed Dependencies [10] and CLEAR Dependencies [7], for which subtle inter-system discrepancies exist. Universal Dependencies (UD) initiative seeks to resolve this issue by globally integrating inconsistent annotation schemes utilized by different dependency representation systems into a coherent and universal format appropriate for multiple languages [27]. The benefit of such a uniform syntactic representation points to the support for cross-lingual learning experiments and the ability to perform comparative analysis, which has already begun to yield meaningful results in parsing and POS tagging for both resource-poor and resource-rich languages [1, 20].

2.1.4.2 Deep Dependency Graph

Choi [4] introduced a dependency graph structure known as Deep Dependency Graph (DDG) that preserves only two of the tree properties, *single-root* and *connected*. In contrast to previous efforts to represent dependency graphs, DDG features primary and secondary dependency arcs, where the primary dependencies correspond to the dependencies found in a conventional dependency tree. Additionally, the secondary dependencies are introduced to capture relations missed by a dependency tree due to the constraints of *single-head* and *acyclic* properties. While increasing the complexity of the dataset, the secondary dependencies greatly enrich the amount of information found in the converted dependency graphs. Finally, semantic roles available in the original phrase structure corpora in the form of function tags are stored on the head nodes of the corresponding dependency relation.

DDG is an effort to represent deep structures [8], and therefore seeks to construct the same dependency graph for phrases or sentences that carry similar meaning but have different surface forms. This involves a complete representation of predicate argument structures (Section 2.1.5) from which the primary meaning of the sentence originates, making DDG a great resource for semantic role labeling discussed in the next section.

2.1.5 Predicate Argument Structure

A predicate describes something about a subject, such as its state or an action performed at or by the subject. Hence, English grammar often associates a predicate with a verb.

Consider the following two sentences:

1. *Michael played the guitar.*

2. *Sam was awake by 9 a.m.*

In Sentence (1), the predicate corresponds to the verb, *played*. However, Sentence (2) features a non-verbal predicate with copula, *was*, that signals that what follows, the adjective *awake*, will describe the state of the subject. Since the verb *was* adds little meaning to the sentence, different views exist on whether the copula should be recognized as a predicate or not. Unlike UD, DDG considers the adjective *awake* to be the predicate. In fact, a preposition phrase (whose head may be an adjective, adverb or other POS) may also become a predicate in DDG.

But a predicate requires arguments to complete its meaning. The predicate *played* in Sentence (1) is helped by two argument: *Michael* is the agent of the action, while *the guitar* is the patient of the action[1]. The predicate *awake* in Sentence (2) is also helped by two arguments: *Sam* is the agent whose state is defined by the predicate, while *by 9 a.m.* provides a temporal setting. In fact, the analysis of locating the predicate, identifying its arguments and specifying the predicate argument relation is known as semantic role labeling, also called shallow semantic parsing.

[1]Consider the passive construction: The guitar is played by Michael.

follow.03:be subsequent	ARG0:causal agent	ARG1:thing followed up on
follow.05:logical conclusion	ARG1:conclusion	ARG2:thesis
follow.07:fulfill	ARG0:fulfiller	ARG1:promise

Table 2.1: Partial PropBank frameset for the predicate *follow*.
First column specifies the meaning of the predicate through a sense ID and its definition. Second and third columns describe the corresponding rolesets for each sense.

2.1.5.1 PropBank

This thesis adopts semantic role relations established in the Proposition Bank (PropBank) [19], a corpus that annotates semantic roles of arguments that accompany predicates found in phrase structure treebanks such as the Penn Treebank and the OntoNotes corpus. The annotation guideline for each predicate is provided through the predicate's PropBank frameset, which defines a set of sense IDs as well as the argument rolesets for each sense. Table 2.1 shows an example of a partial frameset for the predicate *follow*.

We show the core list of the semantic roles below, along with their prototypical descriptions.

- ARG0: agent
- ARG1: patient
- ARG2: instrument, benefactive, attribute
- ARG3: starting point, benefactive, attribute
- ARG4: ending point
- ARGM: modifier

ARGM can be further modified by a thematic role labels, whose complete list can be found in [25].

2.2 Constituent-to-Dependency Conversion

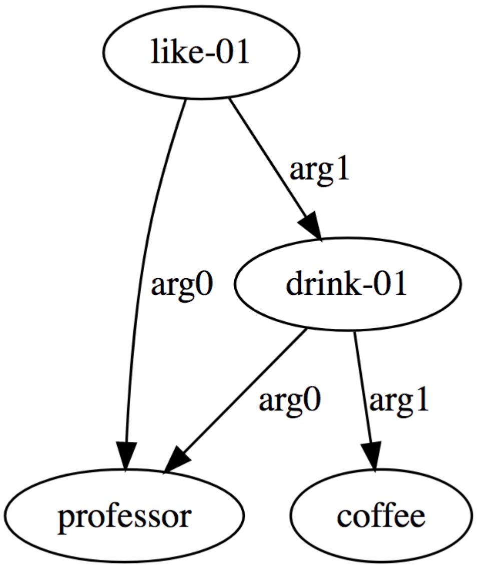
In the domain of syntactic parsing, two types are primarily used for practical applications: constituency and dependency parsing. Historically, constituency parsing has been the dominant approach due to the availability of large annotated phrase structure corpora [16, 25]. In fact, the construction of parsed text in English during the 1990s was what encouraged researchers to experiment with various statistical learning approaches in building efficient and effective parsers.

As for the dependency treebanks, only a few manually annotated corpora are available in English [21]. To build a dependency parser that exploits the statistical learning or the recent neural network approaches, then, one has to manually annotate raw text. However, such labor is not only costly but also time-consuming. A realistic alternative is to convert the existing phrase structure trees into dependency trees through a procedure known as constituent-to-dependency conversion.

Because the constituent-to-dependency conversion is a necessary step for supplying a dependency parser with sufficient training data, it is a well-studied topic in NLP [15, 5]. The conversion requires head-percolation rules, originally introduced by Magerman [15], to establish the head-child dependency relation. It specifies for each phrasal POS (1) the search direction of left or right and (2) a sequence of POS tags (See Table 3.1 for an example of head-percolation rules). For each node encountered, the algorithm searches through the node's immediate children in the direction specified in (1) and compare the POS of each child with the leftmost POS among the sequence provided in (2) that has not been checked already. When a match is found, the chosen child becomes the head of the siblings, while the rest are made dependent of the head.

Once the dependency structure is established, the POS and dependency relations have to be inferred. This is done through linguistic heuristics, relying on morphological analysis, POS and word forms of the head and the dependent, as well as function tags if they are available. Our source code used for both the Penn Korean Treebank and the Kaist Treebank is available at: `https://github.com/emorynlp/ud-korean`.

```
(l / like-01
     :arg0 (p / professor)
     :arg1 (d / drink-01
         :arg0 p
         :arg1 (c / coffee)))
```
(a) PENMAN-based AMR notation



(b) Graph notation

Figure 2.5: Example AMR annotation of a sentence "*The professor likes to drink coffee.*"

In the domain of Korean NLP, Choi and Palmer [6] has carried out dependency corpus generation in Korean which was subsequently used to train and test a statistical parser. Choi [3] went on to prepare Korean data for SPMRL (Statistical Parsing of Morphlogically Rich Languages) 2013 shared task, for which the Kaist Treebank [22] was converted into dependency trees. Chun et al. [9] presented three dependency treebanks in Korean derived from existing treebanks to follow the latest UD annotation guidelines, whose conversion methodology constitutes the first part of this thesis.

Finally, it is worth mentioning that there has been a separate effort to compile a UD style dependency treebank in Korean, known as Hani corpora. However, the corresponding published exposition is not available at this time.

2.3 Abstract Meaning Representation Parsing

Abstract Meaning Representation (AMR) [2] is a semantic representation language that expresses the meaning of the sentence as a rooted, directed, labeled graph, as shown in Figure 2.5. A culmination of previous efforts on creating a semantic treebank (sembank) that features what had previously existed as separate annotations, such as named entities and co-reference, AMR is expected to spur further semantic parsing research as the Penn Treebank had done for constituency parsing.

Nodes in AMR feature semantic *concepts*, *variables* and/or *relations*. For instance, consider the following snippet extracted from the first two nodes in Figure 2.5a: `(l / like-01: arg0 (p / professor))`. `(p / professor)` uses a variable `p` to denote an instance from the concept `professor`. An immediate benefit of introducing variables becomes clear when handling coreference; the variable previously defined can re-enter into the graph at a later point. The relation `arg0` is a familiar notion from PropBank; it describes the semantic role played by the variable `p` in its relation to the predicate `like`. Note that the predicate is followed by an integer, `(l / like-1)`, where the integer refers to the word's sense ID found in PropBank frameset. Since AMR seeks to abstract away from syntactic idiosyncrasies, sentences or phrases with similar meaning will feature the same AMR.

The AMR parsing, then, is the task of discovering AMR structure within raw text. Two approaches have proven to be effective so far. The first approach first maps a span of words to a concept. Then, given a sequence of span of words and a sequence of concepts, the model learns a scoring function for all possible relations between identified concepts. The model outputs the final AMR by finding a maximum spanning graph, i.e. a graph that maximizes the sum of the scores [11, 26].

The second approach exploits the apparent similarity between dependency structure and AMR: the head-child dependency. It first runs a dependency parser to obtain a dependency tree of the source text. Then, a transition-based framework transforms the input dependency tree into its corresponding AMR [24]. An extension of the work further enriched the model training procedure by adding additional linguistic features such as coreference as input to the mapping framework, obtaining better results [23].

Chapter 3

Approach

3.1 Dependency Conversion in Korean

3.1.1 Google UD Korean Treebank

McDonald et al. [17] prepared the first version of Korean UD Treebank, one of six treebanks released in the work, by scraping approximately 6K sentences from newswire and weblogs. The sentences were subsequently tokenized, POS-tagged and annotated with dependency relations. However, the UD annotation guidelines at the time differed significantly from the current version of the universal guidelines, the Universal Dependencies version 2 (UDv2). In this section we describe our approach in automatically converting the original trees to meet the current guidelines. Each step of conversion other than POS-relabeing is accompanied by a figure demonstrating the result of conversion on a sample tree given in Figure 3.1. The Google UD Korean Treebank (GKT) was distributed as a

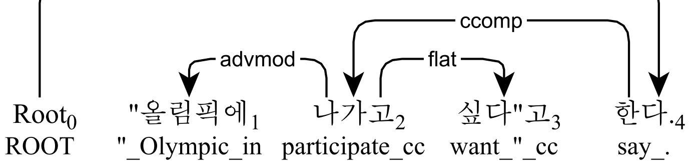


Figure 3.1: Example dependency tree based on dev-s909 from the original GKT

part of the CoNLL'17 shared task datasets.

3.1.1.1 Morphological Analysis

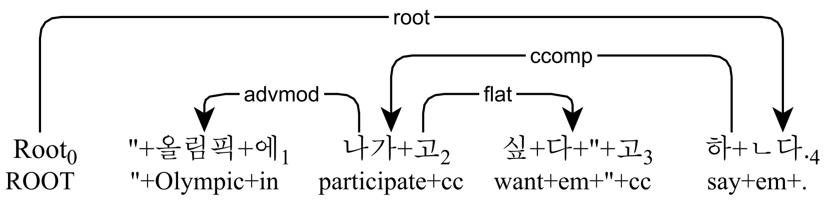


Figure 3.2: After morphological analysis

The authors of the original GKT consider the automatic tokenization carried out for the corpus to be generally too coarse-grained; the suffixes, particles, punctuation marks and/or other symbols were left in with the tokens [17]. While the authors call for a manual correction by the annotators, the corpus had remained unchanged.

To help remedy this issue, we first augment the corpus with the morphological analysis obtained by an automatic morphological analyzer, the KOMA tagger [14], which uses a morpheme tagset found in the Sejong Treebank [13]. Figure 3.2 shows the morpholgical analysis obtained for the original tree in Figure 3.1. The morphological analysis is included in the last column of the data.

3.1.1.2 Re-Tokenization

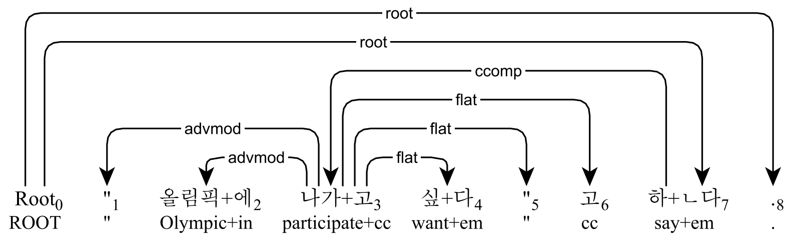


Figure 3.3: After re-tokenization

Based on the morphological analysis thus obtained, we proceed to address the coarse tokenization issue. Specifically, we address the the re-tokenization of punctuation marks and symbols to correctly configure the dependency relations[1]. Figure 3.3 shows the 1st and 3rd tokens in Figure 3.2 are re-tokenized, as they originally contained a double quote. Over 9K tokens with embedded punctuation are revised, resulting in 3K additional tokens after re-tokenization.

3.1.1.3 POS Re-Labeling

Given the re-tokenized corpus, we update the original POS to match the current POS tagset endorsed in UDv2. Additionally, measures are taken to assign appropriate POS to separated tokens in the previous section based on the morpheme tags. Note that the original GKT provides two POS tags for each token (columns 4 and 5). Our relabeling focuses on replacing the first set of POS tags in column 4, and for the sake of consistency with other corpora, the second POS column is removed from the corpus.

3.1.1.4 Head ID Re-Mapping

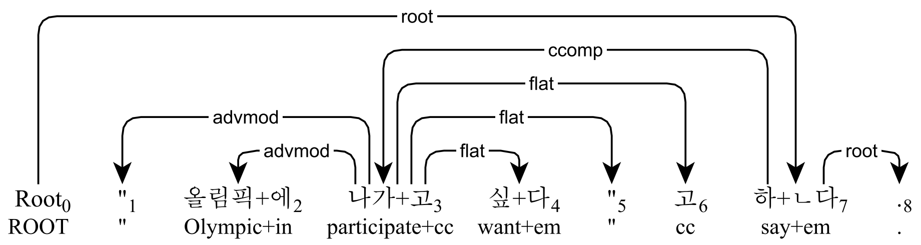


Figure 3.4: After head ID re-mapping

As a consquence of re-tokenization, the head IDs of the separated tokens must be redirected. In general, the word inherits the original head ID, while the punctuation points to the previous token, i.e. token from which the punctuation was split, as seen with the 8th token in Figure 3.4. An exception is made for quotations or parenthetical phrases. Based on the observation that in general a quotation forms a sentence, a quotation (marked by quotation marks (“ ”), as seen with the 1st and 3rd tokens in Figure 3.2) will feature its own sub-dependency tree where only its root will link to an element outside of the

[1]A complete re-tokenization of particles is beyond the scope of this study.

quotation. Therefore, the root of the sub-dependency tree is located by finding the link from within the quotation to an outside element. Punctuation points to the head of the quotation, as seen with the 1st and 5th tokens in the Figure 3.4.

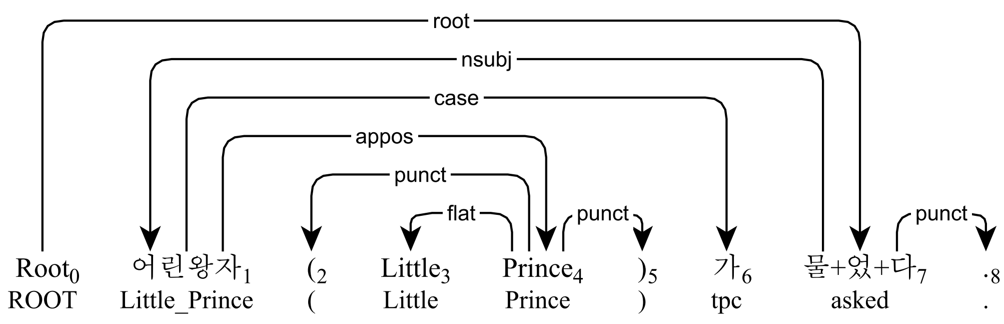


Figure 3.5: Example dependency tree with a prenthetical expression of a sentence "*어린왕자(Little Prince)가 물었다.*"

In the case of parenthetical expressions involving (), <>, [], ‘’ and ≪≫, we found that in the vast majority of cases, the elements within the parenthetical symbols are supplementary phrases describing a preceding token. This being so, the head of the parenthetical phrase is assigned to the rightmost element[2]. When the parenthetical expression originally forms a single token with the preceding word, as seen with the example sentence in Figure 3.5, the token preceding the parenthetical expression inherits the original head ID and becomes the head of the root of the parenthetical expression. If there are any case particles attached to the right of the parenthetical (the 6th token in the same figure), then the case markings are also made dependent on the token preceding the parenthetical expression.

[2]Note that Korean is a head-final language, i.e. the heads of all phrase types typically occur at the rightmost node.

3.1.1.5 Dependency Re-Labeling

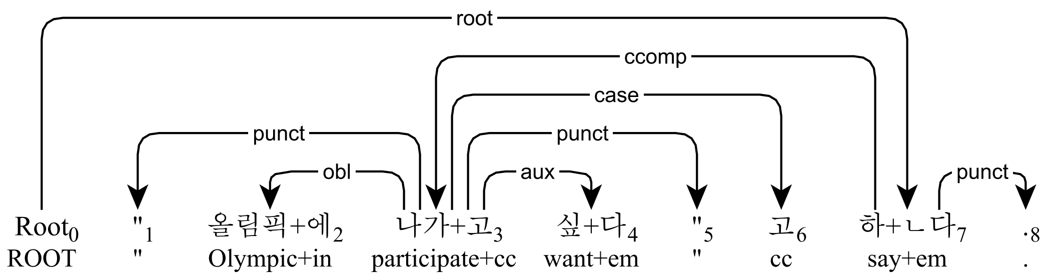


Figure 3.6: After dependency re-labeling

Similar to the POS re-labeling, we update the dependency relations to reflect the latest updates in UDv2. For instance, *Olympics+in* is annotated as an adverbial modifier (`advmod`) of *participate* prior to UDv2, as seen in Figure 3.4. We relabel this as an oblique (`obl`) relation in our corpus, as specified in UDv2. Additionally, we infer dependency relations for the tokens that were tokenized during the re-tokenization step through linguistic heuristics using morpheme tags, POS and word forms of the head and the dependent.

3.1.1.6 Lexical Correction

Since a portion of the original GKT was gathered from weblogs, we perform a manual check for spelling errors. We report a total of 146 tokens whose spelling errors are detected and fixed. Finally, HTML entity symbols are replaced with corresponding lexical symbols.

3.1.2 Penn Korean Treebank

The Penn Korean Treebank (PKT) is the first large-scale bracketed corpus in Korean using a phrase structure annotation, drawing on military-domain documents and newswire [12]. At 15K sentences, the corpus is the only treebank in Korean that features empty categories, which enable to represent long-distance dependencies. We exclude the military-domain texts from the corpus due to its lack of generality, reducing the size of the corpus to around 5K sentences.

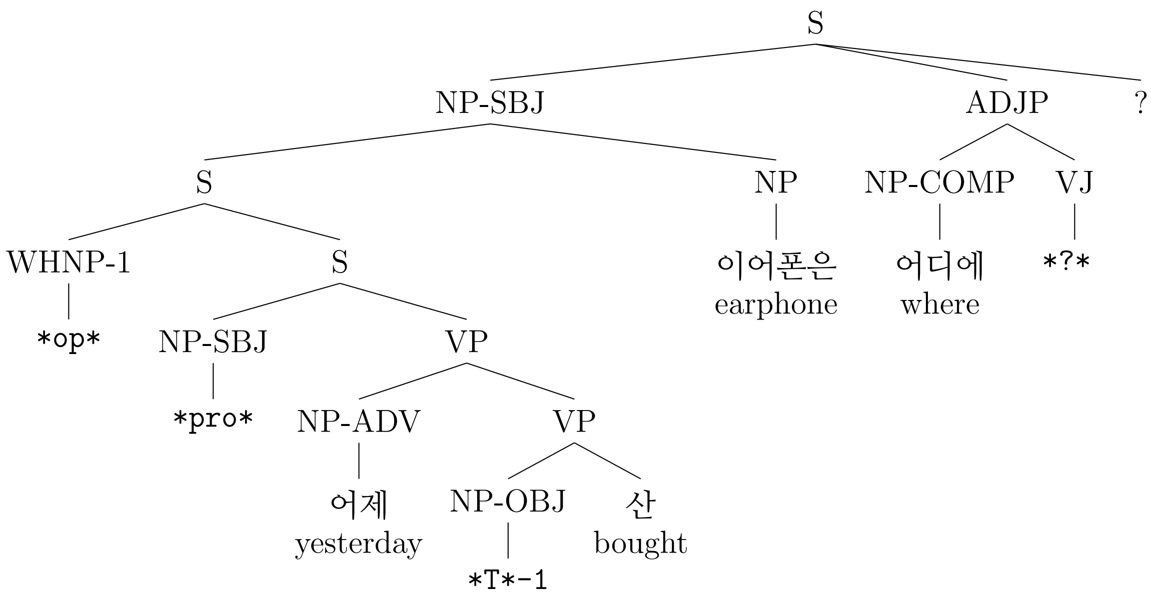


Figure 3.7: Before trace movement
Example of 4 types of empty categories: `*op*`, `*pro*`, `*T*`, `*?*`

3.1.2.1 Mapping Empty Categories

As discussed in Section 2.1.3, PKT features four empty categories. Below we describe our empty category mapping procedure.

- Trace (`*T*`): Given a terminal node that represents a trace like `(NP-OBJ *T*-1)` in Figure 3.7, we locate its antecedent, `(WHNP-1*op*)` which shares the same integer index 1. Then we reorder the tree in such a manner that the subtree with the antecedent's non-terminal node as a root is extracted out of its position and inserted in place of the trace node, resulting in Figure 3.8.

- Ellipsis (`*?*`): While the annotation in PKT does not provide sufficient contextual information to resolve the dropped predicate in a matrix clause, it is possible to map elided elements that share a predicate intra-sententially. First, we locate the shared predicate, then redirect what would have been a dependency between an elided element and its child to a dependency between the child and the node immediately to the left of the shared predicate. Since PKT does not provide an index that links the ellipsis token to its antecedent, we perform simple heuristics of matching POS tags and morpheme tags, as well as function tags if they exist, for dependency

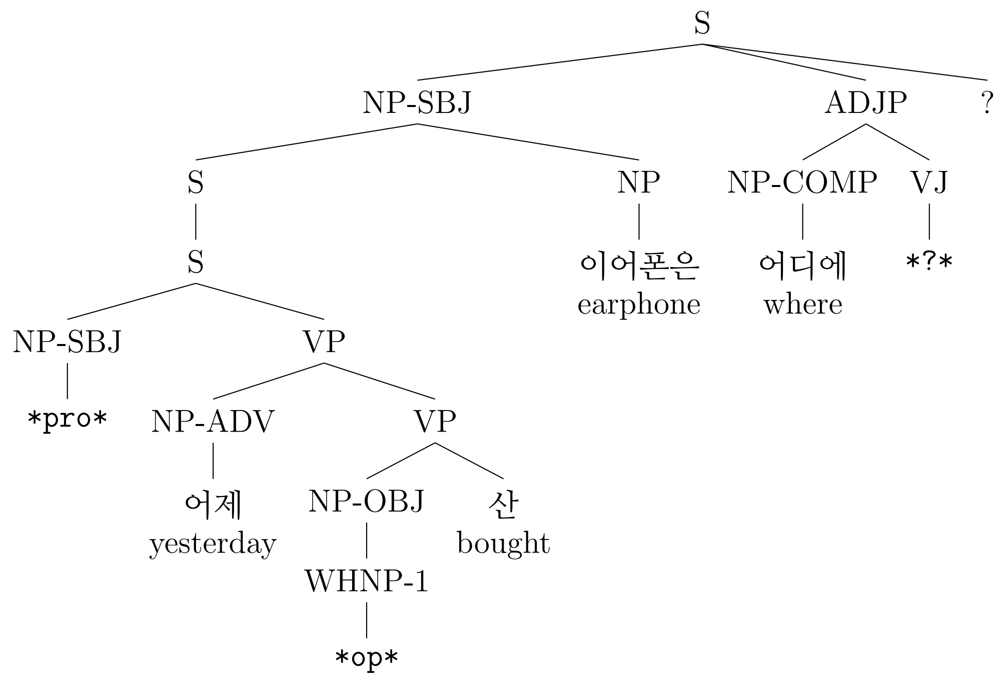


Figure 3.8: After trace movement

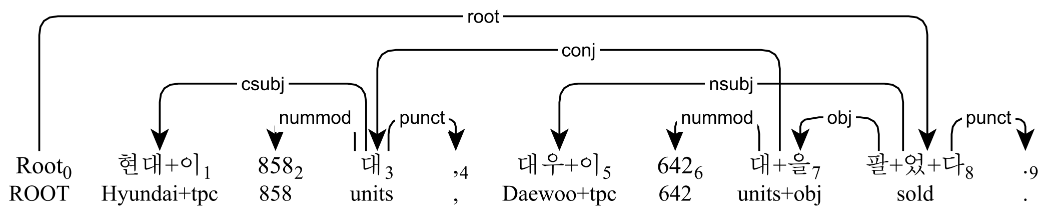


Figure 3.9: Example dependency tree with ellipsis

redirection. We represent this relationship as a fixed conjunct, as shown with the dependency between the 3rd and 7th token in Figure 3.9.

- Empty Assignment (`*pro*`) & Empty Operator (`*op*`):

  No explicit steps are taken to reorder sentence structures with these empty categories; these do not require a sentence reorder for correct interpretation.

3.1.2.2 Coordination

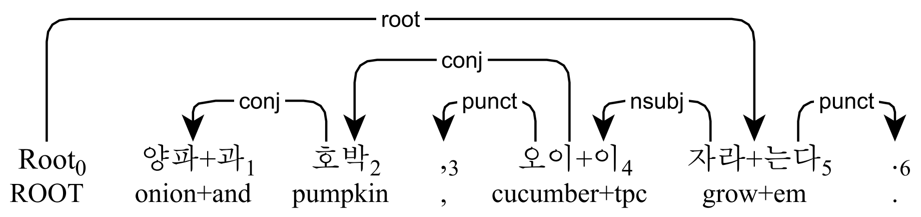


Figure 3.10: Example dependency tree with coordination

Following the guideline of [6], each conjunct points to its right sibling as its head so that the rightmost conjunct becomes the head of the phrase. This aligns with the notion that Korean is a head-final language. Our conversion discovers coordination structure by applying a set of heuristics [9]. An example of the coordination structure is shown in Figure 3.10, where 호박 (*pumpkin*) is the head of its left sibling 양파+와 (*Onion+tpc*), and 오이+이 (*Cucumber+tpc*) is made the head of the entire noun phrase with coordination.

3.1.2.3 POS Tags

We infer POS tags based on the morphological analysis of each token. The mapping is mostly categorical, where the last morpheme often plays a pivotal role in the inference. One exception is `DAN` (determiner-adnominal) which is a superset of (1) demonstrative prenominals (e.g. 이(*this*), 그(*that*)) and (2) attribute adjectives (e.g. 헌(*old*), 새(*new*)). We map the former to `DET` (determiner) and the latter to `ADJ` (adjective).

3.1.2.4 Dependency Relations

The re-labeling of dependency relations starts with handling empty categories, discussed in Section 3.1.2.1. Then each node is assigned its head with head-percolation rules based on Table 3.1. The dependency relationship between the node and its head is inferred by investigating the morphological analysis, POS, word forms of the head and the dependent, as well as function tags if they exist.

Phrase	D	Headrules
S	r	VP;ADJP;S;NP;ADVP;*
VP	r	VP;ADJP;VV|VJ;CV;LV;V*;NP;S;*
NP	r	N*;S;N*;VP;ADJP|ADVP;*
DANP	r	DANP|DAN;VP;*
ADVP	r	ADVP;ADV;-ADV;VP;NP;S;*
ADJP	r	ADJP;VJ;LV;*
ADCP	r	ADC;VP;NP|S;*
ADV	r	VJ;NNC;*
VX	r	V*; NNX;*
VV	r	VV;NNC;VJ;*
VJ	r	VJ;NNC;*
PRN	r	NPR;N*|NP|VP|S|ADJP|ADVP;*
CV	r	VV;*
LV	r	VV;J;*
INTJ	r	INTJ;IJ;VP;*
LST	r	NNU;*
X	r	*

Table 3.1: Headrules for PKT. **Phrase** lists all phrasal tags in PKT. **D** denotes the search direction, and r denotes searching for the drightmost constituent. In the **Headrules** column, * denotes any tag headed by its precedent alphabet, and | denotes logical or. Each **Headrule** gives higher precedence to the left tag on the list.

3.1.3 Kaist Treebank

The Kaist Treebank (KTB) [22] contains approximately 31K phrase structure trees annotated with different bracketing guidelines from PKT. Unlike PKT, KTB does not include empty categories and function tags, but features a more fine-grained morpheme tagset.

3.1.3.1 Coordination

Coordination in KTB is handled in much the same way as described in Section 3.1.2.2 for PKT. Additionally, the lack of empty categories in KTB implies the lack of verb ellipsis annotation. As was the case for PKT, the rightmost conjunct becomes the head of the coordination phrase.

3.1.3.2 POS Tags

The Kaist POS inference is again mostly categorical based on the morphological analysis, with added emphasis placed on the rightmost morpheme.

3.1.3.3 Dependency Relations

KTB dependency conversion follows the procedure outlined for PKT, although the absence of empty categories has made the empty category handling unnecessary. Once the head of nodes is located with head-percolation rules based on Table 3.2, we infer the dependency relations. Unlike PKT, however, the absence of function tags and the small number of phrasal POS pose an additional challenge for the dependency inference task. While the rich morpheme tagset is certainly helpful, it often fails to provide the sufficient contextual information necessary for robust inference. For instance, consider the particle morphemes *jcs*, *jcc*, and *jxt*, which help identify a subject node. In PKT, a function tag `-SBJ` will ensure that the dependency is either `nsubj` or `csubj`, depending on whether the subject is nominative or clausal. However, while *jcs* and *jcc* roughly correspond to `nsubj` or `csubj`, *jxt* only suggests that the node is the topic of the phrase or the clause (which may be `nsubj`, `csubj` or other relation). This suggests that KTB offers no systematic way of distinguishing one dependency relation from another in certain situations.

Phrase	**D**	**Headrules**
S	r	VP;ADJP;S;NP;ADVP;*
VP	r	pv*|pa*|n*|VP|NP;ADJP;S;*
NP	r	n*|f|NP|S|pv*|VP|pa*|ADJP;ADVP|MODP;*
ADJP	r	ADJP|pa*|n*;ADVP;VP;NP;S;*
ADVP	r	ADVP;VP;ma*;NP;S;*
AUXP	r	AUXP;NP;p*;n*;px;*
MODP	r	mm*;VP;ADJP;NP;*
IP	r	ii;p*;n*;ADVP;m*;*
X	r	*

Table 3.2: Headrules for KTB (see Table 3.1 for tabular details)

3.2 Dependency-AMR Parsing

3.2.1 Transfer Learning

As noted by Wang et al. [24], there is a resemblance between dependency structure and AMR: the head-child dependency. Consider Figure 2.4 and Figure 2.5b. With a predicate in the root position, its immediate children serve to fulfill the meaning of the predicate in both figures. For instance, *professor* is `arg0` to the predicate *like* in Figure 2.5b as *girl* is `nsubj` to the predicate *left* in Figure 2.4.

This similarity motivates a direct integration of the PropBank labels into DDG, where the dependency labels between a predicate and its arguments can be completely replaced by the corresponding semantic roles. In the above example, this step would replace the `nsubj` in Figure 2.4 with `arg0`, since *girl* is an agent that performs the action *left*. The product of such integration for all predicates in OntoNotes will yield a new corpus of dependency graphs, each of which represents a dependency structure whose predicate argument relations are labeled with PropBank labels.

The greatest merit of this new corpus is as an additional resource in semantic role labeling. Since AMR inherits the semantic role relations from PropBank, any AMR parsers will benefit from an effective semantic role labeler. To this end, we propose a novel method of AMR parsing by using this dataset to train a dependency parser on dependency parsing and semantic role labeling, which is subsequently trained again—through transfer learning—on the AMR parsing task.

This is not the first time that a dependency parser is used in developing an AMR parser. Recall that in the transition-based framework for AMR parsing [24], the dependency trees produced by a dependency parser are mapped to their corresponding AMRs. Since the mapping procedure requires a separate framework from a parsing model, however, the role of the dependency parser is rather limited.

Our approach differs from this in that, instead of designing an additional model for mapping the output of the dependency parser into AMR, the dependency parser is trained again on the AMR parsing task. It is our expectation that the initial training that exposes the parser to semantic role labeling task will help the model perform AMR parsing.

3.2.2 PropBank Integration into OntoNotes DDG Corpus

```
nw/wsj/17/wsj_1705.parse 19 1 gold be be.01 ----- 0:1-ARG1 1:0-rel 2:1-ARG2
```

Figure 3.11: Sample PropBank entry

Integration of PropBank labels during the constituent-to-dependency conversion of the OntoNotes corpus is generally a straight-forward process. Each tree in the OntoNotes is assigned one or more PropBank entries depending on the number of predicates in the source sentence[3]. By reading each entry, we locate the predicate as well as its arguments specified in the entry. Then, once the conversion algorithm produces a DDG of the phrase structure tree, we find the corresponding dependency in DDG and replace the it with the appropriate PropBank label.

Consider a sample PropBank entry, shown in Figure 3.11. The first column denotes the filename where the corresponding phrase structure tree resides. The second column denotes the index of the tree from the top of the file, starting at 0. The third column denotes the index of the predicate in the tree, starting at 0. These point to the exact location of the predicate, and the sixth column provides the word's sense ID found in the predicate's PropBank frameset.

The right side of the entry shows the locations of arguments. In the sample entry above, the predicate has a total of 2 arguments[4]: `ARG1` and `ARG2`. Each of these PropBank labels are preceded by two integers, delimited by a colon. The first integer corresponds to the index of the argument in the tree. The second integer refers to the 'height' of the argument node. To see the purpose of the height, consider the following phrase structure tree in the OntoNotes corpus, found in the `nw/wsj/17/wsj_1705.parse` file. Note that this is the tree whose predicate argument structure is annotated in the sample PropBank entry above.

```
(TOP (S (NP-SBJ (DT This))
        (VP (VBZ is)
            (NP-PRD (NN football)
                    (NN country)))
        (. .)))
```

Figure 3.12: Sample OntoNotes phrase structure tree (tree index 19 in nw/wsj/17/wsj_1705.parse)

[3]There are files in the OntoNotes corpus that does not have a corresponding PropBank data. We skip these during our conversion.

[4]`rel` refers to the predicate itself, and is therefore not an argument.

The PropBank entry in Figure 3.11 tells us that the tree index of `ARG1` is 0, which points to the `(TOP (S (NP-SBJ (DT This))` in the Figure 3.12. However, it is difficult to tell whether the argument is the determiner `(DT This)`, the noun phrase `(NP-SBJ (DT This))`, or even the entire subtree rooted by `S`. Height enables for an easy disambiguation in such a case; starting at the terminal node with a value of 0, it increases every time one moves up one node in the phrase structure tree. Hence, `(DT This)` has a height of 0, `(NP-SBJ (DT This))` has 1, and so on.

Once we locate the predicate as well as its arguments, we can proceed to identifying the dependency between these nodes and replace the label with the semantic role played by the argument. However, this approach is not always successful. First, there may not be a corresponding DDG dependency. Second, even if there does exist a DDG dependency, naive replacement of labels may not be sufficient. Below we outline two such representative cases that require special handling.

3.2.2.1 Coordination

DDG captures coordination through a combination of a primary dependency and one or more secondary dependencies. Specifically, the primary dependency connects the head of the coordination to the predicate. Then for every sibling of the head, DDG introduces a secondary dependency as a conjunct relation between the shared predicate and each sibling.

However, PropBank entries only give an index to the head of the coordination phrase, making it necessary to infer additional arguments hidden in a coordination sentence or a phrase. We tackle this problem by checking whether the siblings of the indexed argument with a height bigger than 0 has a secondary conjunction dependency to the predicate.

3.2.2.2 Copulas

Generally known as *be* verbs, copulas receive different treatments in PropBank and DDG when it comes to establishing predicate argument structure. Consider again the example sentence *"Sam was awake by 9 a.m."* Whereas DDG assigns the predicate role to *awake*, PropBank does so to *was*, as seen in Section 2.1.5. This situation is not limited to just an adjectival predicate; DDG allows any preposition phrase preceded by a copula to become a predicate. Due therefore to this different interpretation of copula constructions, the conversion algorithm may simply skip a PropBank entry whose predicate cannot be located.

Our approach triggers a different handling mechanism for copulas. Once we detect a PropBank entry with a *be* verb as a predicate, we iterate through the arguments in the PropBank entry. For each argument node, we check if the node's dependency head is also an argument to the predicate *be*, which in theory cannot be because a meaningful dependency between arguments can only be established through the shared predicate. If the condition is true, however, we recognize the dependency head of the argument node as the true predicate and reformulate the dependencies for other arguments to originate from the new predicate.

Chapter 4

Analysis

4.1 Dependency Treebanks in Korean

4.1.1 Corpus Analytics

	GKT	**PKT**	**KTB**	Total
Sentences	6,339	5,010	27,363	38,712
Nodes	80,392	132,041	350,090	562,523

Table 4.1: Treebank statistics for three Korean dependency treebanks

The overall statistics of the phrase structure treebanks used for dependency conversion are shown in Table 4.1. While GKT features a larger number of sentences than PKT, PKT's higher number of nodes per sentence likely reflects the newswire domain of PKT. KTB, with the largest number of sentences, draws on diverse domains ranging from news editorials to academic textbooks and has the sentence complexity of 12 nodes per sentence, which is comparable to that of GKT.

4.1.1.1 POS Analysis

The distribution of POS in the three Korean dependency treebanks are shown in Figure 4.1. The figure shows a relatively consistent proportion for each POS, with the top three

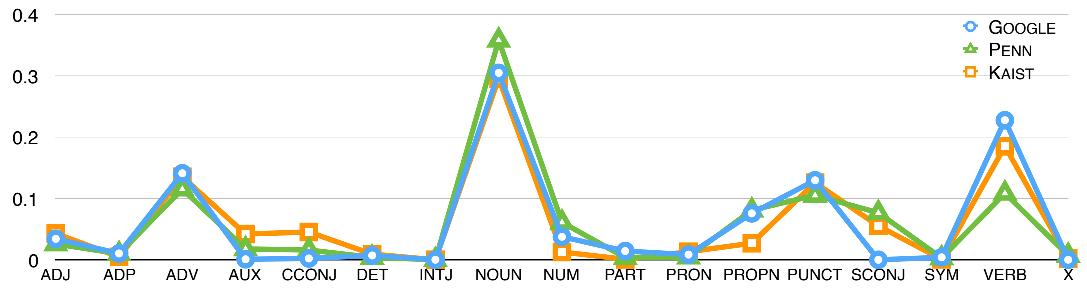


Figure 4.1: POS distribution for three Korean dependency treebanks

most frequent being NOUN, VERB, and ADV, closely followed by the fourth PUNCT. Table 4.2 displays the number of occurrences for all POS tags in the three treebanks. Below we consider several representative cases in which there appears to be a mismatch in proportion among the three corpora.

NOUN and VERB One of the characteristics of Korean newswire articles that comprise PKT is a frequent occurence of nouns to produce a tone that is objective and factual. We suspect that this is the primary cause of PKT's higher proportion of NOUN and lower proportion of VERB relative to other two corpora. Since more nouns appear in PKT than usual, it leads to a high proportion of NOUN. Additionally, since verbs are often neutralized into noun forms in the news domain, it may account for the low proportion of VERB, among other possible reasons.

AUX The initial annotation of GKT chooses to label all verbs, primary or auxiliary, as VERB. This is rather a puzzling annotation guideline, as AUX is a well-established POS in the Korean grammar. While we introduce AUX for some of the tokens affected by re-tokenization, its proportion relative to other POS remains noticeably lower than other corpora.

PART The absence of PART is a remnant of coarse tokenization, where particles remain embedded to its morpheme root in the original GKT. We partially introduce PART for particles attached before or after a punctuation, and therefore split from the morpheme root after re-tokenization of punctuation marks and symbols.

Tag	Description	GKT	PKT	KTB
ADJ	Adjective	2,760	3,431	14,223
ADP	Adposition	1,791	1,251	1,498
ADV	Adverb	11,361	15,174	49,204
AUX	Auxiliary	74	2,263	12,906
CCONJ	Coordinating Conjunction	223	2,453	19,368
DET	Determiner	573	685	4,824
INTJ	Interjection	3	0	56
NOUN	Noun	32,345	46,866	105,193
NUM	Numeral	847	7,931	4,848
PART	Particle	31	464	268
PRON	Pronoun	682	857	7,712
PROPN	Proper Noun	490	12,257	12,366
PUNCT	Punctuation	10,440	13,428	38,925
SCONJ	Subordinating Conjunction	0	9,780	18,466
SYM	Symbol	328	376	260
VERB	Verb	18,431	13,855	59,273
X	Other	13	970	700
Total		80,392	132,041	350,090

Table 4.2: Frequencies of POS in the final resulting Korean dependency corpora

4.1.1.2 Dependency Analysis

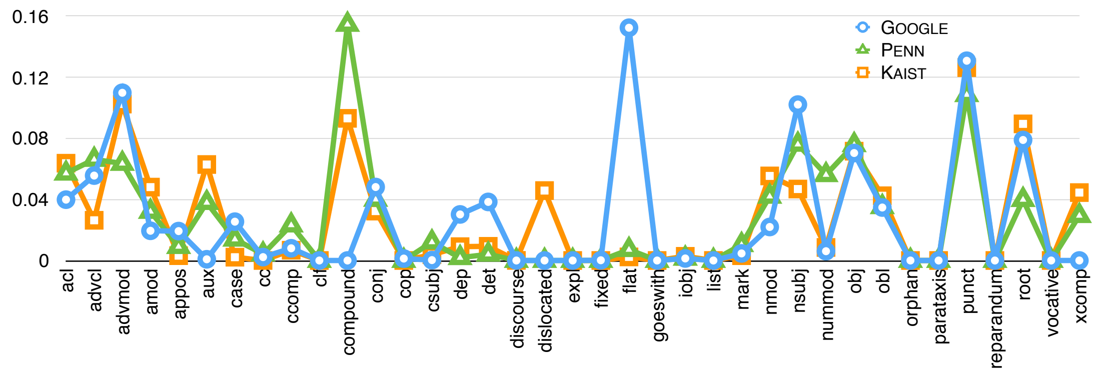


Figure 4.2: Dependency label distribution for three Korean dependency treebanks

Figure 4.2 displays a relative dependency label distribution for the three Korean dependency treebanks. In contrast to POS, there appears to be some inconsistency in the proportion of dependency labels among the three corpora. We pay special attention to some of the representative mismatch cases, listed below.

flat Another by-product of coarse tokenization, `flat` was used by the annotators of GKT during the initial annotation phase to cover cases in which no appropriate dependency label could be identified. Consider the example dependency tree in Figure 3.1, where `flat` describes the dependency between 나가고 (*participate_cc*) and 싶다"고 (*want_"_cc*). Since the second node is a combination of what should have been three different tokens, the dependency of `flat` is assigned between the two nodes. While we address this issue as it arises during re-tokenization, GKT's abundant annoation of `flat` implies that many other tokens require additional dependency re-labeling which is beyond the scope of this work.

compound For a similar reason stated for `flat`, GKT's lack of `compound` can be traced to coarse tokenization; where there should have been a `compound` to represent a multi-word expression, `flat` was instead assigned. The noticeably high proportion of `compound` in PKT likely reflects the news domain, as described in **NOUN and VERB** paragraph in Section 4.1.1.1.

csubj, nsubj and dislocated As noted in Section 3.1.3.3, the many-to-many mapping from subject-related morpheme tagset (*jcs*, *jcc*, *jxt*) to subject-related dependencies (`csubj`, `nsubj`, `dislocated`) in KTB explains the particularly odd label proportion of the labels for KTB.

4.1.2 Remaining Issues

GKT While we address a number of problems in GKT through our re-tokenization and subsequent handling of the revised tokens, there remains some issues that need to be fixed in future updates. Most of the remaining errors again originate from the coarse tokenization, leading to problems such as an incorrect dependency relation. In addition, the head-first analysis of coordination does not align well with Korean that is a head-final language, where the rightmost node—not the leftmost node—is most suited to become the head of the coordination.

Tag	Description	GKT	PKT	KTB
acl	Clausal Modifier of Noun	3,198	1,488	21,468
advcl	Adverbial Clause Modifier	4,515	11,636	20,487
advmod	Adverbial Modifier	8,810	2,964	19,102
amod	Adjectival Modifier	1,566	1,595	16,584
appos	Appositional Modifier	1,544	1,182	1,059
aux	Auxiliary	64	4,807	18,935
case	Case Marking	1,624	1,548	1,343
cc	Coordinating Conjunction	223	785	5,234
ccomp	Clausal Complement	651	9,858	15,655
clf	Classifier	0	0	1
compound	Compound	0	28,908	24,696
conj	Conjunct	3,863	9,960	20,774
cop	Copula	102	418	303
csubj	Clausal Subject	21	8,014	1,202
dep	Unspecified Dependency	2,437	609	3,019
det	Determiner	3,077	685	4,824
discourse	Discourse Element	0	0	47
dislocated	Dislocated Elements	0	0	20,964
expl	Expletive	0	0	0
fixed	Fixed Multiword Expression	13	528	3,186
flat	Flat Multiword Expression	12,252	18	803
goeswith	Goes With	0	0	0
iobj	Indirect Object	108	222	967
list	List	0	0	0
mark	Marker	372	1,003	799
nmod	Nominal Modifier	1,761	5,555	22,045
nsubj	Nominal Subject	8,290	4,012	17,444
nummod	Numeric Modifier	489	154	3,295
obj	Object	5,801	9,823	23,605
obl	Oblique Nominal	2,784	3,357	11,577
orphan	Orphan	0	0	0
parataxis	Parataxis	0	0	0
punct	Punctuation	10,494	13,073	39,016
reparandum	Overridden Disfluency	0	0	0
root	Root	6,332	5,036	27,363
vocative	Vocative	0	0	15
xcomp	Open Clausal Complement	1	4,803	4,278
Total		80,392	132,041	350,090

Table 4.3: Frequencies of dependency labels in the final resulting Korean dependency corpora

PKT and KTB While PKT, known for its strong annotation consistency, offers function tags and well-publicized documentation which has led to an efficient development of a conversion algorithm, this is not the case for KTB. As discussed in Section 3.1.3.3, the lack of function tags and a small phrasal POS tagset provides an additional challenge in inferring a dependency relation, in some cases offering no systemaic way of distinguishing one dependency relation from another.

4.2 PropBank-Augmented OntoNotes DDG Corpus

Analysis of the current version of the PropBank-Augmented OntoNotes DDG corpus begins by analyzing whether the predicate argument relation specified in a PropBank entry has a corresponding dependency in DDG. We consider the following two possibilities:

1. There exists a corresponding dependency. We call this a `match` case.

2. There does not exist a corresponding dependency. We call this a `no-match` case.

Match Case When there is a match case, we replace the dependency label with its corresponding PropBank label. However, caution is necessary as it is possible that the targeted dependency may have been mis-identified due to a different interpretation of some linguistic constructions such as copula (Section 3.2.2.2). For instance, ARG0 should generally describe a subject relation of the argument to the predicate, which corresponds to subject-related dependencies such as `nsbj` in DDG. However, when the identified DDG dependency of ARG0 has, for example, `det` (determiner) label, it would require analysis of annotations in both the DDG and the PropBank entry for the validity of the match case. To help identify these pitfalls, we record the dependency labels for all `match` cases before replacement.

No-Match Case When no corresponding dependency can be identified, we record the count of `no-match` cases for a given PropBank label and store the DDG for future analysis. We suspect that the primary cause of the no-match case is similar to what was described above: different methods of handling certain linguistic constructions by DDG and PropBank.

Label	**Top 1**	**Top 2**	**Top 3**	**Total**	**No-Match %**
ARG0	`nsubj`	no-match	`r-nsubj`	206,087	12.5%
	159,474	25,721	8,472		
ARG1	`obj`	`nsubj`	no-match	318,132	16.2%
	120,130	66,403	51,553		
ARG2	no-match	`ppmod`	`obj`	97,910	48.0%
	47,000	23,428	7,507		
ARG3	`ppmod`	no-match	`obj`	6,916	13.2%
	3,897	914	563		
ARG4	`ppmod`	`adv`	no-match	5,302	3.4%
	4,037	747	182		
ARG5	`adv`	`ppmod`	`prt`	132	0.0%
	66	35	23		
Total	no-match			634,479	**19.8%**
	125,370				

Table 4.4: Top 3 dependency label distribution of `match` and `no-match` cases for each numbered argument in PropBank

Table 4.4 displays the statistics of `match` and `no-match` cases for the numbered arguments. Compared to other labels, ARG2 shows a disportionately large percentage of `no-match` cases, at around 50%. Overall, we report the overall percentage of the `no-match` case at 19.8%, an improvement on the initial percentage at 26.8%. In the next section we describe remaining challenges that we hope will further reduce the `no-match` proportion in the corpus.

4.2.1 Remaining Challenges

Below we describe two representative challenges whose solutions are yet to be implemented.

Non-Verbal Predicates Two primary constructions are usually responsible for non-verbal predicates: copulas and light verbs. While our method of handling copulas is described in Section 3.2.2.2, we speculate that there exists a generalized solution of our method that can be applied to handle all non-verbal predicates, including copulas, semi-copulas (e.g. become, get, etc) and light verbs.

A light verb adds little semantic contribution to the meaning of the predicate. Instead, light verbs are followed by a noun which acts as a verb and describes the subject of the predicate, such as *grading* and *call* in *do the grading* and *give a call.* In PropBank, a small portion of entries with a noun as a predicate signals a light verb construction.

It has been suggested that there exists a noticeable discrepancy between how DDG and PropBank recognize light verbs. While DDG admits various light verb constructions which contain a verb from a set of light verbs *make, take, have, do, give, keep* and a noun from a set of 2,474 eventive nouns, PropBank is more prudent in making such a decision and often assigns the predicate role to the verb in the multi-word expression. This discrepancy can lead to a mismatch between PropBank that identifies the verb as a predicate and DDG that identifies the eventive noun as a predicate.

We perform a simple check in order to detect inconsistency between DDG and PropBank for both cases of non-verbal predicates, copulas and light verbs. For every PropBank entry, before attempting to replace any dependency labels, we iterate through every argument listed in the PropBank entry and test whether there exists a dependency between the argument and other arguments of the predicate. Theoretically this is impossible, as a meaningful semantic and dependency relation between arguments can only be established through the predicate. However, as seen with cases like light verbs, PropBank often assigns a predicate role to what is considered either a copula or one of the arguments of the predicate in DDG's interpretation, which would satisfy the above condition. In this case we identify the head of the PropBank's predicate as the true predicate, and this aligns with the DDG interpretation of the non-verbal predicates. Afterwards we redirect other arguments to originate from the new predicate. Note that this is a generalized version of our approach in handling copulas in Section 3.2.2.2, where we adopt a similar procedure for handling PropBank entries with *be* verb as a predicate.

Gerunds A gerund, characterized by the *-ing* form attached to the end of the verb root, functions as a noun in a compound noun phrase. For example, *supporting banks* is a

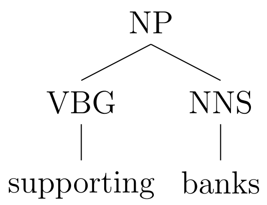


Figure 4.3: Example of a gerund in a phrase "*supporting banks*"

noun phrase in which the verb *support* modifies the noun *banks* as a gerund (Figure 4.3). PropBank handles gerunds by identifying *support* as the predicate and *banks* as ARG0. However, *banks* is the head of the entire noun phrase in dependency representation. Therefore, the corresponding dependency in DDG is not between the two terminal nodes (`VBG` supporting) and (`NNS` banks), as is specified in PropBank, but rather between (`VBG` supporting) and (`NP` (`VBG` supporting) (`NNS` banks)).

Chapter 5

Conclusion

In this work, we outline the conversion procedure for constructing (1) three dependency treebanks in Korean and (2) a PropBank-Augmented OntoNotes DDG corpus. Each of the three Korean dependency treebanks is derived from its original structure to follow the latest UD guidelines, UDv2. The Google Korean Treebank is the only corpus that originally contained dependency trees, albeit with coarse tokenization and outdated annotations, which are systematically revised and updated. The Penn Korean Treebank and the Kaist Treebank are converted from phrase structure to dependency. A total of 38K+ dependency trees are generated. To the best of our knowledge, this is the first time that the three Korean treebanks are converted into dependency treebanks under the latest UD annotation guidelines, resulting in a large and consistent dependency treebank.

The public release of this data has an immediate impact on facilitating further research in dependency parsing in Korean, where the lack of training data has remained an obstacle. Furthermore, researchers analyzing other morphologically rich languages can benefit from an additional resource for a comparative analysis of their parsing systems. Finally, we expect that the conversion methodologies from phrase structure to dependency will serve as a helpful reference to those wishing to carry out the same conversion for other corpora.

Future directions include further enhancements to the quality of the treebanks and the development of Korean dependency parser trained on this dataset. It will be interesting to see the impact of the trained dependency parser on other Korean NLP applications. We make our automatic conversion source code available at: `https://github.com/emorynlp/ud-korean`.

In the second part of the thesis, we introduce an on-going project for creating a new corpus of DDG converted from the OntoNotes phrase structure trees with semantic features found in PropBank and AMR. Since AMR originates from PropBank, AMR parsers will benefit from any work that improves on the performance of a semantic role labeler. To this end, this work leverages the dependency graph structure of DDG that captures the full predicate argument structure by integrating PropBank labels during the constituent-to-dependency conversion of the OntoNotes corpus. This is a preliminary step before a full integration with additional semantic features in AMR.

Moving forward, we will continue to analyze the `no-match` cases to identify and address the inconsistency in annotation between DDG and PropBank. It is our expectation that the final version of this large-scale corpus derived from OntoNotes will represent the complete predicate argument structure with dependency arcs labeled with PropBank semantic roles, along with rich syntactic information stored in primary and secondary dependencies of DDG. We hope that this new corpus will serve as a helpful resource for dependency parsing as well as semantic role labeling.

Once this preliminary step is complete, we plan to experiment with transfer learning of a dependency parser trained on the dataset for the AMR parsing. The initial results of this parsing model can serve as a meaningful baseline for future experiments. Additionally, we will move on to the next phase of the research, where we carry out a full integration of AMR features into the corpus.

Bibliography

[1] Alonso, H. M., Agic, Z., Plank, B., and Søgaard, A. Parsing universal dependencies without training. *CoRR abs/1701.03163* (2017).

[2] Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. Abstract meaning representation for sembanking, 2013.

[3] Choi, J. D. Preparing Korean Data for the Shared Task on Parsing Morphologically Rich Languages. Tech. Rep. 1309.1649, ArXiv, 2013.

[4] Choi, J. D. Deep Dependency Graph Conversion in English. In *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories* (Bloomington, IN, 2017), TLT'17, pp. 35–62.

[5] Choi, J. D., and Palmer, M. Robust Constituent-to-Dependency Conversion for English. In *Proceedings of the 9th International Workshop on Treebanks and Linguistic Theories* (Tartu, Estonia, 2010), TLT'10, pp. 55–66.

[6] Choi, J. D., and Palmer, M. Statistical Dependency Parsing in Korean: From Corpus Generation To Automatic Parsing. In *Proceedings of the IWPT Workshop on Statistical Parsing of Morphologically Rich Languages* (Dublin, Ireland, 2011), SPMRL'11, pp. 1–11.

[7] Choi, J. D., and Palmer, M. Guidelines for the Clear Style Constituent to Dependency Conversion. Tech. Rep. 01-12, University of Colorado Boulder, 2012.

[8] Chomsky, N. *Lectures in Government and Binding.* Dordrecht, Foris, 1981.

[9] Chun, J., Han, N.-R., Hwang, J. D., and Choi, J. D. Building Universal Dependency Treebanks in Korean. In *Proceedings of the 11th International Conference on Language Resources and Evaluation* (Miyazaki, Japan, 2018), LREC'18.

[10] de Marneffe, M.-C., and Manning, C. D. Stanford typed dependencies manual, sep 2008.

[11] Flanigan, J., Thomson, S., Carbonell, J., Dyer, C., and Smith, N. A. A discriminative graph-based parser for the abstract meaning representation.

[12] Han, C., Han, N., Ko, E., and Palmer, M. Development and evaluation of a korean treebank and its application to NLP. In *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain* (2002).

[13] Hong, Y. 21st Century Sejong Project Results and Tasks (21세기 세종 계획 사업 성과 및 과제). In *New Korean Life (새국어생활)*. National Institute of Korean Language, 2009.

[14] Lee, D.-G., Rim, H.-C., and Lim, H.-S. A syllable based word recognition model for korean noun extraction. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1* (Stroudsburg, PA, USA, 2003), ACL '03, Association for Computational Linguistics, pp. 471–478.

[15] Magerman, D. M. Natural language parsing as statistical pattern recognition. *CoRR abs/cmp-lg/9405009* (1994).

[16] Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology* (Stroudsburg, PA, USA, 1994), HLT '94, Association for Computational Linguistics, pp. 114–119.

[17] MCDONALD, R., NIVRE, J., QUIRMBACH-BRUNDAGE, Y., GOLDBERG, Y., DAS, D., GANCHEV, K., HALL, K., PETROV, S., ZHANG, H., TÄCKSTRÖM, O., BEDINI, C., BERTOMEU, N., AND LEE, C. J. Universal dependency annotation for multilingual parsing. In *In Proc. of ACL '13* (2013).

[18] NIVRE, J., AND SCHOLZ, M. Deterministic dependency parsing of english text. In *Proceedings of the 20th International Conference on Computational Linguistics* (Stroudsburg, PA, USA, 2004), COLING '04, Association for Computational Linguistics.

[19] PALMER, M., GILDEA, D., AND KINGSBURY, P. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics 31*, 1 (2005), 71–106.

[20] PLANK, B., SØGAARD, A., AND GOLDBERG, Y. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *CoRR abs/1604.05529* (2016).

[21] RAMBOW, O., CRESWELL, R., SZEKELY, R., TABER, H., AND WALKER, M. A dependency treebank for english. In *In Proceedings of the 3rd International Conference on Language Resources and Evaluation, Las Palmas, Gran Canaria* (2002).

[22] SUN CHOI, K., HAN, Y. S., HAN, Y. G., AND KWON, O. W. Kaist tree bank project for korean: Present and future development. In *In Proceedings of the International Workshop on Sharable Natural Language Resources* (1994), pp. 7–14.

[23] WANG, C., XUE, N., AND PRADHAN, S. Boosting transition-based amr parsing with refined actions and auxiliary analyzers. In *ACL* (2015).

[24] WANG, C., XUE, N., AND PRADHAN, S. A transition-based algorithm for amr parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015), Association for Computational Linguistics, pp. 366–375.

[25] Weischedel, R., Hovy, E., Marcus, M., Palmer, M., Belvin, R., Pradhan, S., Ramshaw, L., and Xue, N. Ontonotes: A large training corpus for enhanced processing, 01 2011.

[26] Werling, K., Angeli, G., and Manning, C. D. Robust subgraph generation improves abstract meaning representation parsing. *CoRR abs/1506.03139* (2015).

[27] Zeman, D., Popel, M., Straka, M., Hajic, J., Nivre, J., Ginter, F., Luotolahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinkova, S., Hajic jr., J., Hlavacova, J., Kettnerová, V., Uresova, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C. D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., dePaiva, V., Droganova, K., Martínez Alonso, H., Çöltekin, c., Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonca, G., Lando, T., Nitisaroj, R., and Li, J. Conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* (Vancouver, Canada, August 2017), Association for Computational Linguistics, pp. 1–19.