대량의 한국어 구문 트리 태깅 코퍼스 구축을 위한 구문 트리 태깅 워크벤치의 설계 및 구현

장병규, 이공주, 김길창
한국과학기술원, 전산학과

Design and Implementation of Tree Tagging Workbench To Build a Large Tree Tagged Corpus of Korean

Byung Gyu Chang, Kong Joo Lee, Gil Chang Kim
Dept. of Computer Science, KAIST

요약

한국어 구문 트리 태깅 코퍼스는 한국어 문장의 구문 구조가 구문 트리 형태로 태깅된 코퍼스이다. 코퍼스 구축은 사람(annotator)에 의하여 수작업으로 이루어지므로, 많은 시간과 인력을 소모하는 작업이다. 그렇기 때문에 코퍼스 구축을 도와주는 구문 트리 태깅 워크벤치는 코퍼스 구축에 필수적인 요소이다. 본 논문에서는 대량의 구문 트리 태깅 코퍼스를 일관되고 빠르게 구축하기 위한 워크벤치 설계시의 고려 사항을 제시한다. 이러한 고려 사항을 기반으로, 다소 정확한 부분에 대한 태깅만을 수행하는 부분 구문 분석, 태깅한 결과에 대한 검증 과정인 일관성 검사, 편한 구문 트리 태깅을 고려한 사용자 인터페이스, 플랫폼 독립적인 구현 등과 같은 워크벤치의 실제 구현에 대하여 설명한다. 또한, 구문 트리 태깅 워크벤치의 앞으로의 연구 방향을 제시한다.

1. 서론

최근의 자연언어 처리 분야에서는 대량의 코퍼스로부터 추출한 다양한 언어 지식(linguistic knowledge)을 이용하여 여러 문제점들을 해결하려는 노력이 증가하고 있다. 그에 따라, 언어 지식을 추출하기 위한 대량의 코퍼스 구축을 위한 작업도 늘어나고 있다.

한국어 구문 트리 태깅 코퍼스는 한국어 문장의 구문 구조를 구문 트리 형태로 표현하고 있는 코퍼스이다. 문장의 구문 구조는 문장 내의 단어들이 서로 어떠한 관계로 연결되어 있는가를 명시한다. 이러한 정보를 담고 있는 대량의 구문 트리 태깅 코퍼스는 기본적인 자연언어 처리 전반에 걸친 주요한 정보원이 된다. 또한, 한국어 음성 인식이나 문자 인식, 정보 검색 등의 연구 분야에까지 중요한 연구 자료가 된다.

구문 트리 태깅 코퍼스 구축은 기본적으로 사람(annotator)에 의하여 반자동 내지는 수작업으로 이루어진다. 그렇기 때문에, 대량의 코퍼스 구축에는 많은 시간과 인력이 들어가게 되므로, 태깅 작업을 돕기 위한 워크벤치가 필요하다.

본 논문에서는 우선 구문 트리 태깅 워크벤치의 목적과 설계시의 고려 사항에 대하여 언급한다. 이러한 고려 사항에 따라서, 부분 구문 분석, 일관성 검사 등의 기능이 포함되고, 사용자 인터페이스가 설계된 구문 트리 태깅 워크벤치의 설계에 대하여 설명한다. 실제로 구현된 워크벤치에 대하여 기술하며, 부분 구문 분석의 효용성 실험 결과 등을 포함한 몇가지 사항에 관하여 고찰한다. 마지막으로 결론을 맺는다.

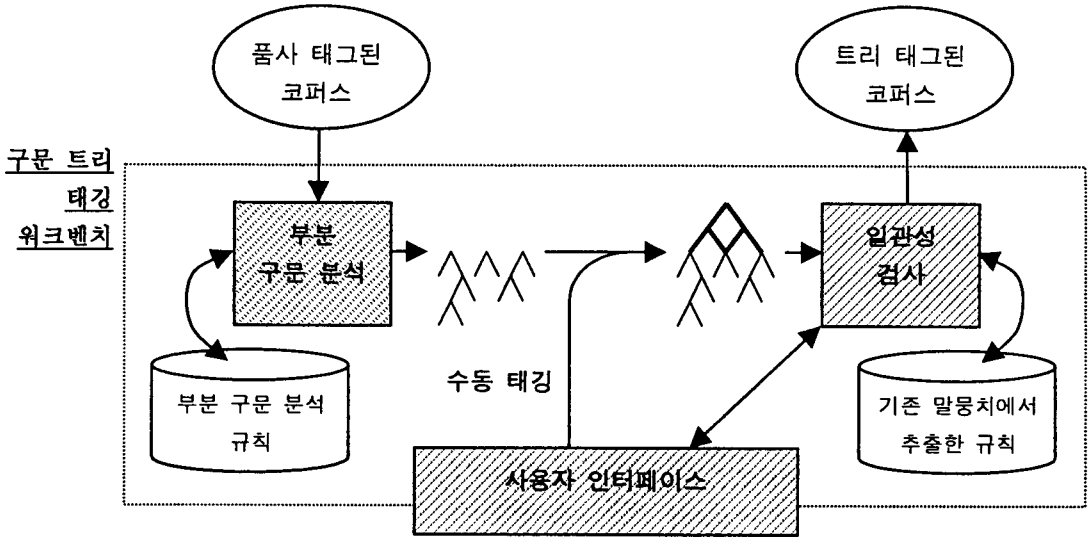


그림 1. 구문 트리 태깅 워크벤치 전체 구성도

2. 구문 트리 태깅 워크벤치의 목적 및 설계시 고려 사항

구문 트리 태깅 워크벤치의 목적은 품사 태깅된 문장을 입력으로 받아서 태깅 수행자(annotator)가 구문 트리 태깅된 문장을 만드는 작업을 도와주는 것이다. 구문 트리를 표현하는 방식에는 여러가지가 있으나[2,3,4], 본 워크벤치에서는 구구조 문법(phrase structure grammar)을 이용하여 구문 트리를 태깅한다. 또한, 문장의 심층적 구조가 아닌 표층적 구조만을 태깅하는 것을 목적으로 한다.

본 논문에서 설명하는 구문 트리 태깅 워크벤치는 다음과 같은 고려 사항에 부합되도록 설계되고 구현되었다. 대량의 구문 트리 태깅 코퍼스의 일관되고 빠른 구축을 위해서 워크벤치 설계시에 고려된 사항은 다음과 같다.

첫째, 대량의 코퍼스 구축을 위해서, 태깅 수행자의 작업양을 줄이면서 태깅 속도를 높이기 위한 방안을 고안해야 한다. 이를 위하여 본 워크벤치에서는 다소 정확하다고 판단되는 일부분을 자동 태깅하는 방안을 고려했다. 또한, 태깅 속도를 향상시키기 위한 사용자 인터페이스를 설계하였다.

둘째, 양질의 코퍼스 구축을 위해서, 태깅 작업의 일관성이 유지되어야 한다. 이를 위하여, 태깅 가이드를 태깅 수행자가 충분히 이해하는 과정이 우선 필요하다. 본 워크벤치에서는 태깅 결과에서 오류 가능성이 있는 부분을 판단하여 다시 태깅하도록 유도함으로써, 일관성 유지를 돕는 방안을 고려하였다.

셋째, 구문 트리 태깅 작업을 용이하고, 편하게 수행할 수 있어야 한다. 대량의 코퍼스 구축 시에는 사람의 장시간 태깅이 필연적이다. 이러한 장시간 태깅 작업을 편하게, 용이하게 돕기 위한 사용자 인터페이스를 고려하였다.

넷째, 태깅 수행자가 많아질수록 여러 플랫폼을 사용할 수 있는 가능성이 높아지므로 플랫폼에 독립적인 구현이 요구된다.

3. 구문 트리 태깅 워크벤치의 설계

본 논문에서 설계된 구문 트리 태깅 워크벤치의 구성은 그림 1과 같다. 구문 트리 태깅 워크벤치의 입력은 형태소 단위의 품사 태깅된 문장이며, 출력은 구문 트리 태깅된 문장이다.

형태소 단위로 품사 태깅된 문장이 들어오면, 다소 정확한 부분에 대한 부분 구문 분석을 통하여 트리 태깅의 일부분을 자동적으로 수행한다. 이렇게 부분 구문 분석된 결과를 사람이

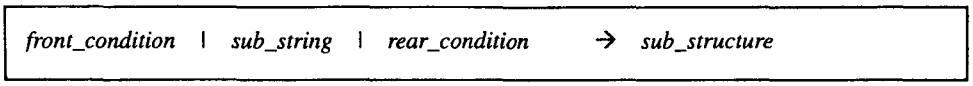


그림 2. 부분 구문 분석기가 사용하는 규칙의 형태

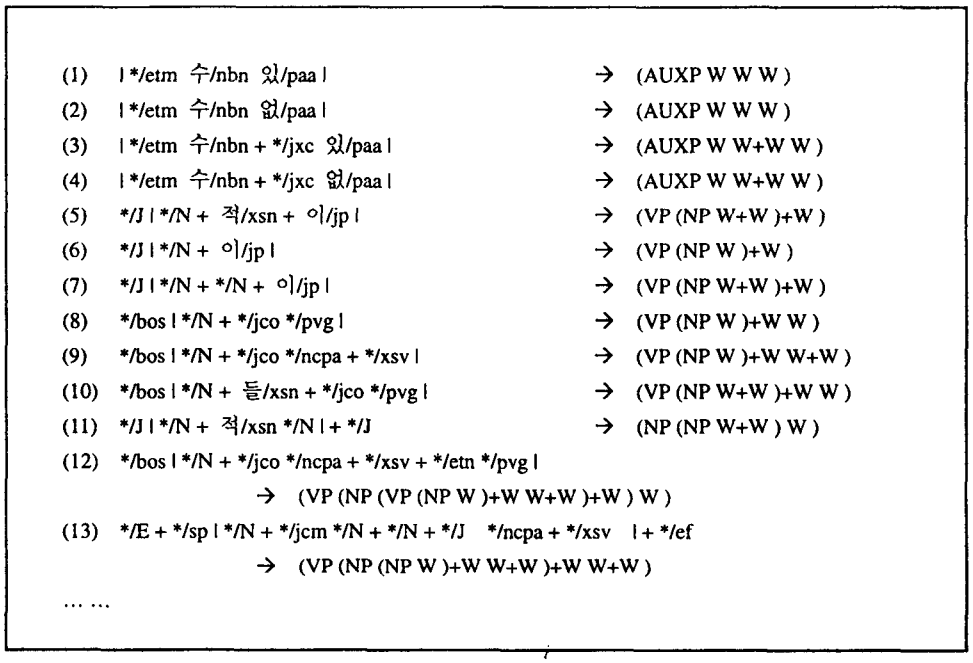


그림 3. 부분 구문 분석을 위한 규칙 (일부)

수동 태깅하여 완전한 구문 트리를 만든다. 한 문장의 태깅이 모두 완료된 이후에 일관성 검사를 통하여 태깅의 오류 가능성을 검출한다. 만약, 오류 가능성이 있는 경우에는 태깅 수행자에게 다시 확인하도록 유도한다. 이러한 과정을 모두 거치면 트리 태깅된 문장이 출력으로 나온다.

다음에서는 자동적으로 트리 태깅의 일부분을 수행하는 부분 구문 분석, 기존의 트리 태깅된 코퍼스에서 추출된 규칙을 이용하는 일관성 검사, 워크벤치의 목적에 부합하도록 설계된 사용자 인터페이스 등의 설계상의 특징에 대하여 자세히 기술한다.

3.1. 부분 구문 분석(partial parsing)

구문 트리 태깅에 소요되는 시간과 구문 트리의 재괄호 치기(re-bracketting) 과정의 빈도수가 적을수록 구문 트리 태깅 작업량이 줄어들면서 태깅 속도가 높아진다[5]. 그러므로, 빠른 태깅을 위하여 트리 태깅을 자동으로 수행하는 구문 분석기는 다음과 같은 특징을 가져야 한다.

첫째, 구문 태깅 도구로서의 구문 분석기는 정확히 하나의 분석 결과만을 제시하여야 한다. 일반적으로 여러 개의 분석 결과로부터 한 개의 정확한 결과를 찾는 것보다는 하나의 분석 결과를 정확한 결과가 되도록 수정하는 것이 경제적이기 때문이다[5].

둘째, 구문 트리 태깅 도구로서의 구문 분석기는 견고해야 한다. 입력 문장의 오류나 시스템의 문법 부족으로 구문 분석의 결과가 나오지 않는 경우에도 부분적인 분석 결과를 제시하여, 사람이 적절한 조치를 취하도록 해야 한다.

이러한 특징들을 고려하여, 본 워크벤치의 구문 분석기는 다소 정확한 부분만의 분석 결과를 제시하는 부분 구문 분석(partial parsing)[8]을 수행한다. 부분 구문 분석은 모호성이 적은 부분에 대한 분석만을 수행하고 모호성을 많이 유발시키는 구조는 분석하지 않음으로써, 결정적으로

```
for POS tagged sentence W = W1 W2 W3...Wn
current_position = 1 ;
while current_position is not end of sentence
    i = current_position ;
    search the rule r that cover the longest
      substring Wi,Wi+1,Wi+2...Wi+j ;
    if rule r is found then
        apply rule r into the substring Wi ...Wi+j ;
        set current_position = i+j+1 ;
    else
        set current_position = i+1 ;
```

그림 4. 부분 구문 분석 알고리즘

구문 분석을 수행하는 방법이다. 입력 문장에 대한 부분 구문 분석의 결과는 여러 개의 부분 구문 트리가 된다.

본 워크벤치에서의 사용하는 부분 구문 분석은 규칙 기반의 분석을 수행한다. 규칙의 형태는 그림 2 와 같으며, *front_condition* 과 *rear_condition* 이 만족되어질 경우, 입력 문장의 *sub_string* 을 *sub_structure* 로 분석하라는 의미이다. 그림 3 은 워크벤치에서 실제로 사용하는 '어휘/품사' 형태로 기술된 규칙의 일부이다. '*'는 모든 어휘, 'N'은 체언류, 'J'는 조사류, 'E'는 어미류를 의미한다.[1] 또한 *sub_structure* 에서 'W'는 *sub_string* 에서의 하나의 형태소를 의미한다. 규칙 (1)에서 (4)번까지는 보조용언구절을 형성하는 규칙이다.

A. 그+는 사실+적 묘사+를 장기로 한다.

B. 그+는 (NP (NP 사실+적) 묘사)+를 장기로 한다.

위의 예와 같이 (A)가 (B)로 분석된 것은, 앞, 뒤에 조사류(J)가 발생할 경우, "명사류+적 명사류"를 "(NP (NP W+W) W)"의 구조로 분석하라는 (11)번 규칙 때문이다. 현재 위와 같은 부분 구문 분석을 위한 규칙은 전문가에 의해 기술되고 있다. 이를 이용한 부분 구문 분석 과정은 그림 4 와 같이 진행된다. 품사 태깅된 문장이 들어왔을 경우에 이 과정이 적용되며, 알고리즘에서 알 수 있듯이 최장 일치법(longest match)을 이용하여 결정적으로 이루어질 수 있다.

[1] 'bos'는 'begin of sentence'이며, 그 외의 사용되는 기호는 [1,2]를 참조하기 바란다.

3.2. 일관성 검사(consistency check)

태깅된 결과들이 일관성을 가지는 양질의 코퍼스 구축을 위해서, 워크벤치에서는 입력 문장의 태깅 결과에 대하여 일관성 검사를 수행한다. 워크벤치에서 태깅 수행자에 의해 현재 태깅된 문장과 기존에 이미 트리 태깅된 코퍼스의 문장들과의 간단한 비교를 통하여 오류를 추출한다. 세밀한 일관성 검사가 쉽지 않기 때문에, 본 워크벤치에서는 태깅된 문장의 간단한 오류 검사 과정으로 설계되었다.

태깅된 결과에 따라서, 태깅된 코퍼스에서 추출되는 구문 규칙들이 달라진다. 예를 들어, 그림 5(b)에서는 굵은 선과 같은 태깅 결과 때문에 "NP → NP+jca NP"인 규칙이 추출된다. 일관성 검사는 이러한 점을 이용한다.

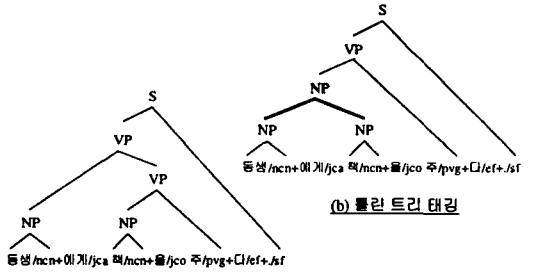


그림 5. 일관성 검사가 이루어지는 문장

NP	→ ncn
VP	→ VP + AUXP
S	→ VP + ef + sf
AUXP	→ ep
VP	→ NP + jco pvg
VP	→ NP + jca pvg
NP	→ NP + jcm ncn
VP	→ NP + jca VP
NP	→ NP + jcj NP + sp maj NP
...	

그림 6. 일관성 검사에 쓰이는 규칙 (일부)

일관성 검사에서 이용하는 규칙들은, 코퍼스에서 추출될 수 있는 규칙들 중에서 사용자에 의해 맞다고 검증된 것들이다. 본 워크벤치에서 설계된 일관성 검사는, 현재 태깅한 문장에서 추출한 규칙이 그림 6과 같은 검증된 규칙 집합에 없다면 오류일 가능성이 있으므로, 사용자가 한 번 검증 과정을 거치도록 하는 것이다. 예를 들어, 그림 5(b)에 있는 굵은 선에서 추출되는 "NP → NP+jca NP" 규칙은 그림 6에 없다. 오류일 가능성이 있으며 실제로 오류이다.

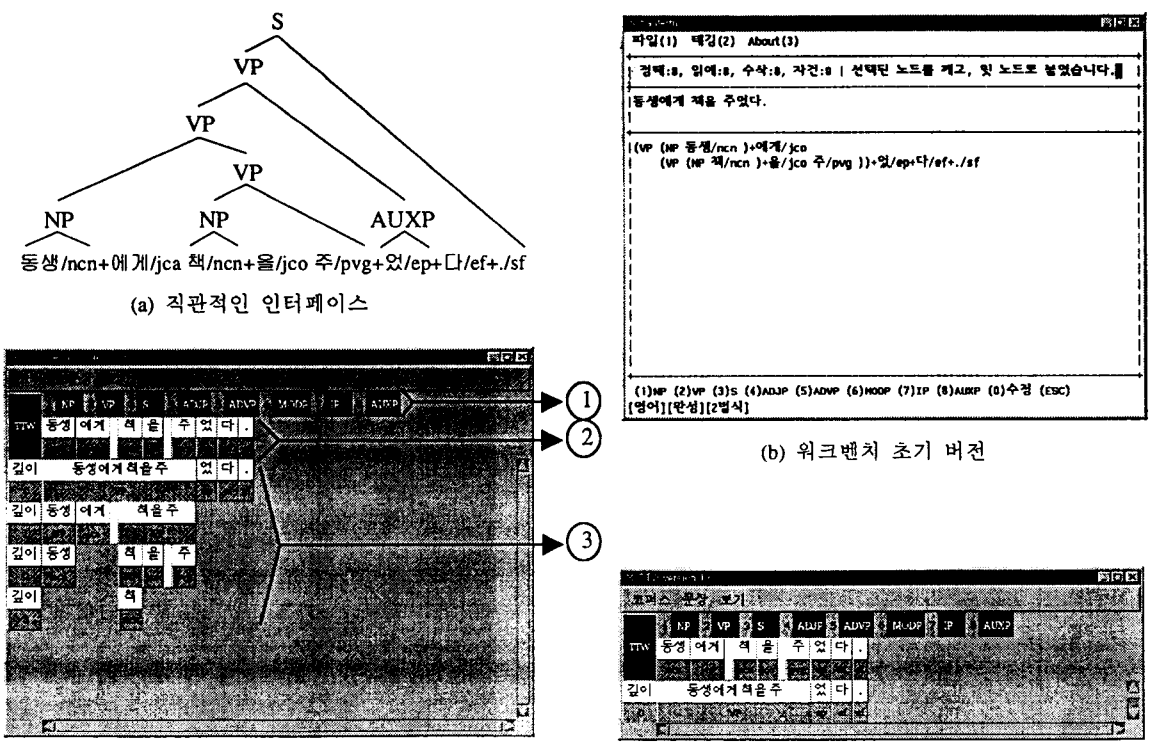
일관성 검사를 위해 검증된 규칙 집합을 얻는 과정은 다음과 같다.

1. 우선 코퍼스에서 일정 빈도 이상의 규칙을 추출하여 사용자에게 확인을 받는다. 한번 검증된 규칙을 초기 규칙들로 이용한다.
2. 일관성 검사 수행 중에 기존의 검증된 규칙들에 없기 때문에 오류일 가능성이 있다고 판단된 규칙이지만 사용자가 옳다고 판단하면, 이를 검증된 규칙에 추가한다.
3. 2번 과정을 반복하여 검증된 규칙의 수를 지속적으로 늘려나간다.

그림 6은 3만 여 문장을 태깅하는 동안 위와 같은 방식으로 구축된, 일관성 검사를 위한 규칙들의 일부분이다. 이러한 방식으로 모든 일관성을 검사할 수는 없지만, 검증되지 않은 규칙들로 태깅된 문장의 경우에는 사용자가 다시 확인하도록 하여 정확성을 높일 수 있다.

3.3. 사용자 인터페이스의 설계

트리 태깅 속도를 향상시키고, 장시간의 트리 태깅 시의 편리성을 도모하기 위하여 사용자 인터페이스 설계에 각별히 주의해야 한다. 특히 트리 태깅되어 있는 문장을 나타내는 방식의 결



(a) 직관적인 인터페이스

(b) 워크벤치 초기 버전

(c) 자바로 구현된 워크벤치 현재 버전 (전체 모드)

(d) 자바로 구현된 워크벤치 현재 버전 (부분 모드)

그림 7. 워크벤치의 사용자 인터페이스

정이 중요하다. 이러한 방식의 차이가 태깅 속도 향상 및 편의성에 가장 큰 영향을 주기 때문이다. 그림 7 은 본 워크벤치 설계시에 고려된 세가지 인터페이스이다.

3.3.1. 트리 자체를 보여주는 방식

우선 (a)와 같이 트리 자체를 보여주는 방식은 일단 초보자들이 쉽게 이해할 수 있으며, 직관적으로 생각할 수 있는 인터페이스이다. 하지만, 긴 문장의 경우 한 화면에 트리가 모두 표현되지 못하는 단점이 있다.

3.3.2. 태깅된 문장을 그대로 보여주는 방식

이러한 이유 때문에, 트리 태깅 워크벤치의 초기 버전은 (b)와 같은 방식으로 구현되었다. 코퍼스가 파일에 저장되는 형태로 표현된 방식으로, 넌터미날(non-terminal)의 깊이(depth)에 따라서 적절하게 탭을 두어 태깅된 상태를 쉽게 나타낸 방식이다. 태깅 수행자가 익숙해지는데 시간이 조금 걸리지만, 텍스트 버전이므로 익숙하게 되면 빠르다는 장점이 있다. 실제로, 3 만 여 문장을 태깅하는데 사용되었으며, 워크벤치에서 필요한 기능들에 대한 구현과 고려가 이 버전에서 거의 모두 이루어졌다. 하지만, 현재 태깅되어야 할 대상뿐만 아니라 여러 형태소들이 함께 나오므로, 조금만 주의를 떨어뜨리면 태깅할 부분이 쉽게 보이지 않는 단점이 있었다.

3.3.3. 현재 태깅할 부분만을 보여주는 방식

이를 해결하기 위하여, 트리 태깅 자체는 괄호 묶기(bracketing)라는 관점에서 (c), (d)와 같은 현재의 인터페이스를 고안하였다. (c)는 전체 모드로 현재까지 태깅된 모든 부분이 보이고, (d)는 부분 모드로 현재 태깅할 대상들만 한 줄에 표시된다. 태깅 시에는 주로 부분 모드를 이용한다.

(1)로 표시된 부분은 부분 트리 구문 태깅 시에 선택할 8 개의 넌터미날의 종류이고, (2)는 원문이며, (3)은 현재까지 태깅된 트리를 나타낸다. 깊이 2 의 "동생/ncn"이 깊이 1 의 "동생/NP"로 묶여 있는 것을 알 수 있다.

(d)와 같은 부분 모드에서는 동일한 태깅 상태가 한 줄에 훨씬 간단히 나타나고 있다. 사용자는 단지 한번의 괄호, 즉, 어디부터 어디까지를 묶을 것인가만을 신경쓰면 되므로, 훨씬 편안하게 빠른 속도로 태깅을 수행할 수 있다. 특히, 한번 묶은 괄호의 경우에는 깨는 경우가 별로 없다는 경험으로도 이러한 인터페이스 방식은 상당한 장점을 지니고 있다고 판단된다. 현재 태깅할 대상뿐만 아니라 전체적인 구조를 알기 위해서는 태깅된 문장을 보는 방식을 (c)와 같은 전체 모드로 바꾸면 된다. 물론, 이 상태에서도 태깅이 가능하다.

편안한 태깅을 위해서, 오른손과 왼손으로 할 작업을 완전히 분리하였다. 오른손은 마우스에서 뗄 필요가 없도록, 부분 트리 구문 태깅 범위를 선택할 때에만 이용한다. 왼손은 키보드에서 뗄 필요가 없도록, 선택한 범위의 넌터미날 종류를 선택할 때(1~8 키)와 문장의 저장, 삭제 등의 기능(s, d 키)에 이용한다. 마우스로 넌티미날의 종류를 선택한다면, 마우스를 움직여야 하므로 태깅 속도가 늦어지며 불편하다. 또한, 넌터미날의 종류 선택을 위한 1~8 키는 선택 횟수가 많은 종류를 1 번 쪽으로 배열하여 왼손의 움직임도 줄였다.

4. 구문 트리 태깅 워크벤치의 구현

현재 워크벤치는 [1]의 품사 태그 집합과 [2]의 구문 트리 표현 양식을 사용해서 구현되었다. 또한, 코퍼스 파일 포맷을 정하였으며, 부분 구문 분석과 일관성 검사를 위한 규칙을 구축하였다. 워크벤치에서 구문 트리 태깅에 관련된 모든 일을 수행하기 위한 기본 기능들을 플랫폼에 독립적으로 구현하였다.

4.1. 코퍼스 파일

워크벤치에서 사용하는 코퍼스 파일은 그림 8 과 같은 포맷으로 되어 있다. 그림 8 의 주석문

부분은 포맷에 대한 간단한 설명이다. 그림 8 은 두 문장이 완전히 태깅된 상태로 저장되어 있으나, 품사 태깅된 문장이나 트리 태깅 중인 문장도 동일한 포맷으로 저장된다.

4.2. 구축된 규칙

워크벤치에서 현재 사용되는 부분 구문 분석을 위한 규칙은 모두 387 개로, 전문가에 의하여 기술되었다. 약 3 만 여 문장을 트리 태깅한 후, 사용자가 일관성 검사를 위해서 검증한 규칙의 수는 모두 1,386 개이다.

4.3. 기본 기능

구문 트리 태깅 워크벤치에는 다음과 같은 기본적인 기능들이 있다.

- 부분 트리의 구문 태깅과 수정: 구문 트리 태깅 작업의 기본적인 괄호치기 기능을 수행하기 위하여, 괄호를 묶고 깨는 기능을 구현하였다.
- 문장을 파일에 저장 혹은 삭제: 태깅 도중의 중간 결과나 완전하게 태깅된 문장을 저장하거나, 적절치 못한 입력 문장을 삭제하기 위한 기능을 구현하였다.
- 입력 문장의 수정: 입력으로 들어오는 품사 태깅된 문장에는 품사 태깅 오류가 있을 수 있다. 또한, 원문 자체에도 오류가 있을 수 있다. 그렇기 때문에, 입력으로 들어오는 문

```
#  1. 처음에 '#'이 나오는 라인은 주석문이다.
#  2. 처음에 ';'이 나오는 라인은 트리 태깅된 문장의 원문이다.
#     트리 태깅된 문장에서 원문을 정확히 얻어내기가 힘들기 때문에 원문이 필요하다.
#     (한국어는 첨가어이므로, "어두운 ← 어둡/pvg+운/etm"과 같은 복구가 쉽지 않다.)
#  3. 트리 태깅 문장은 공백 라인으로 구분된다.
#  4. 트리 태깅 문장은 몇 라인에 걸쳐 나와도 된다.
; 우리가 이웃인 까닭은 가까이 있음이 아니고 따뜻한 정을 나눔입니다.
(S
   (VP
      (NP
         (VP (NP 우리/npp )+가/jcs
            (VP (NP 이웃/ncpa )+이/jp ))+ㄴ/etm 까닭/ncn )+은/jxt
      (VP
         (ADJP
            (NP
               (ADJP (ADVP 가까이/mag ) 있/paa )+ㅁ/etn )+이/jcs
            아니/paa )+고/ecc
         (VP
            (NP
               (VP
                  (NP (ADJP 따뜻하/paa )+ㄴ/etm
                     정/ncn )+을/jco 나누/pvg )+ㅁ/etn )+이/jp )))+ㅂ니다/ef+./sf )

; 대도시의 도로는 이제 포화상태입니다.
(S
   (VP (NP (NP 대도시/ncn )+의/jcm 도로/ncn )+는/jxt
      (VP (ADVP 이제/mag )
         (VP (NP 포화상태/ncn )+이/jp )))+ㅂ니다/ef+./sf )
```

그림 8. 트리 태깅 코퍼스 파일의 예 (일부)

장의 오류를 수정하기 위하여 간단한 편집기를 제공하였다.

- 결과를 구문 트리 형태로 보여주는 기능: 구문 트리가 태깅된 결과를 그림 7(a)와 같은 구문 트리 형태로 보여주는 기능을 구현하였다.

4.4. 플랫폼에 독립적인 구현

대량의 코퍼스 구축 시에는 태깅 수행자가 여러 이종의 컴퓨터를 사용하여 태깅하는 경우가 많게 된다. 이러한 플랫폼의 문제를 해결하기 위하여, 자바 언어를 이용하여 워크벤치를 구현하였다. 이를 통해서, 태깅 수행자가 익숙한 어떠한 플랫폼에서도 사용할 수 있다. 현재까지 자바 언어로 만든 응용 프로그램이 빠른 속도를 내지는 못하지만, 태깅 워크벤치에는 빠른 속도가 요구되는 부분이 별로 없기 때문에 문제가 되지 않을 것으로 예상된다.

5. 토의 및 고찰

5.1. 부분 구문 분석의 효용성

부분 구문 분석의 효용성을 알아보기 위하여, 완전하게 트리 태깅된 31,086 문장을 이용하여 실험을 수행해 보았다. 우선 31,086 문장에 대해, 부분 구문 분석만을 자동으로 수행하였다. 그리고, 부분 구문 분석만 된 결과를 완전하게 트리 태깅된 결과와 비교하여 정확률과 재현율을 구하였다[11]. 부분 구문 분석은 이러한 실험에서 정확률 96.8%, 재현율 30.7%를 나타내었다. 즉, 부분 구문 분석기는 매우 정확하게 분석하면서 10 개의 괄호 묶기(bracketing) 중에서 3 개 정도를 자동으로 수행하였다는 뜻으로, 부분 구문 분석의 효용성을 나타낸다.

부분 구문 분석이 다소 정확한 부분만을 분석하는 방식이므로 정확률이 상당히 높은 것은 당연하다. 앞으로 높은 정확률을 그대로 유지하면서 더욱 높은 재현율을 얻는 방법이 연구되어야 한다. 특히, 현재 부분 구문 분석기는 품사 태깅된 문장이 들어오면 부분 분석 규칙을 한번만 적용한다. 하지만, 태깅하는 중간중간에 태깅이 명백한 부분이 생기는 경우가 있으므로, 태깅 중간의 자동 태깅 가능성을 검사하는 방법론이 요구된다. 또한, 현재 전문가가 기술하는 규칙을 자동적으로 획득하는 방안과 부분 분석 규칙을 코퍼스가 구축됨에 따라서 점진적으로 학습하는 방법에 대한 연구가 필요하다.

5.2. 통합 워크벤치

품사 태깅 오류 등을 수정하기 위하여, 워크벤치에는 간단한 편집기를 제공하였다. 편집기에서 문장의 원문 혹은 품사 태깅의 오류를 수정하고, 다시 태깅하는 방식으로 동작한다. 하지만, 이러한 오류를 수정하기 위해서는 구문 트리 태깅 워크벤치를 품사 태깅 워크벤치와 함께 통합하는 것이 좋을 것으로 생각된다. 오류 중에는 하나의 문장에 그치는 것이 아니라 코퍼스 파일 전반에 걸친 오류도 있으며, 품사 태깅 워크벤치가 품사 태깅의 오류를 수정하는데 훨씬 많은 도움을 주도록 설계, 구현되었을 것이기 때문이다. 따라서, 앞으로 통합 워크벤치의 설계와 구현에 대한 연구가 있어야 할 것으로 생각된다.

5.3. 온라인 태깅 가이드

특정한 문장이 이전에 어떠한 방식으로 태깅되었는지를 찾아보는 일이 태깅 도중에 자주 일어난다. 현재는 태깅 가이드를 문서 형태로 만들었지만, 태깅하는 도중에 문서를 찾는 일은 태깅의 효율을 저하시킨다. 이러한 문제를 해결하기 위해서, 워크벤치에는 이전의 용례를 검색할 수 있는 기능이 마련되어야 한다. 이전에 태깅된 코퍼스로 용례 데이터베이스를 구축할 수도 있을 것이며, 코퍼스 파일에서 트리를 검색할 수 있는 프로그램을 만들 수도 있을 것이다. 용례 외에도 온라인에서 여러 가지 태깅 가이드에 관련된 정보를 제공할 수 있도록 연구와 개발이 진행되어야 한다.

6. 결론

본 논문에서는 구문 트리 태깅 워크벤치의 주요 고려 사항을 수동 태깅 작업양의 감소와 태깅 속도의 향상, 태깅 결과의 일관성 유지, 편안한 태깅 작업, 플랫폼에 독립적인 구현과 같은 네가지로 정리하였다.

이러한 목적에 부합되는 워크벤치를 만들기 위하여, 다소 정확한 부분에 대한 태깅만을 수행하는 부분 구문 분석, 이전의 태깅 결과와의 일관성 검사를 통한 태깅 결과의 검증 과정, 태깅 속도 향상과 편안한 태깅을 위한 사용자 인터페이스의 설계 등의 특성을 가지는 워크벤치를 설계하였다. 또한, 이러한 특징을 포함하면서 기본적인 기능을 모두 갖춘 워크벤치를 플랫폼에 독립적으로 구현하였다.

부분 구문 분석의 경우에는, 30.7% 정도의 재현율을 보임으로써 어느 정도의 효용성을 보였으나, 수동 태깅 작업양을 줄이기 위하여 더욱 많은 연구가 수행되어야 할 것으로 생각된다. 또한, 통합 워크벤치와 온라인 태깅 가이드에 대한 연구도 있어야 할 것이다.

참고 문헌

[1] 최기선, 남영준, 김진규, 한영균, 박석문, 김진수, 이춘택, 김덕봉, 김재훈, 최병진, "한국어정보베이스를 위한 형태, 통사 태그 표준에 관한 연구", *인지과학*, Vol.7, No.4, pp.43-61, 1996

[2] 이공주, 김재훈, 최기선, 김길창, "구문 트리 부착 코퍼스 구축을 위한 한국어 구문 태그", *인지과학*, Vol.7, No.4, pp.7-24, 1996

[3] 이공주, 장병규, 김길창, "한국어 구문 트리 태깅 코퍼스 작성 요령", 한국과학기술원 전산학과, CS/TR-97-112, 1997

[4] 이공주, 김재훈, 장병규, 최기선, 김길창, "한국어 구문 트리 태깅 코퍼스 작성을 위한 한국어 구문 태그", 한국과학기술원 전산학과, CS/TR-97-102, 1996

[5] Mitchell P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English : the Penn Treebank", *Computational Linguistics*, Vol.19, No.2, pp.313-330, 1993

[6] Roger Garside, "The Large-Scale Production of Syntactically Analysed Corpora", *Literary and Linguistic Computing*, Vol.8, No.1, pp.39-46, 1993

[7] Geoffrey Sampson, *English for the Computer: The SUSANNE Corpus and Analytic Scheme*, Oxford University Press, 1995

[8] Steven Abney, "Partial Parsing via Finite-State Cascades", *Proc. of the ESSLLI '96 Robust Parsing Workshop*, pp.275-282, 1996

[9] Sean P. Engelson and Ido Dagan, "Minimizing Manual Annotation Cost In Supervised Training From Corpora", *Proc. of ACL '96*, pp.319-326, 1996

[10] David Carter, "The TreeBanker: a Tool for Supervised Training of Parsed Corpora", *Proc. of ACL Workshop: Computational Environments for Grammar Development and Linguistic Engineering* (Madrid), 1997

[11] E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini and T. Strzalkowski, "A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars", *Proc. of Fourth DARPA Speech and Natural Language Workshop*, pp. 306-311, 1991