2 단계 방법을 사용한 빠른 한국어 TAG 구문분석기 구현

박정열
파리 7 대학, 언어학과 - 형식 언어학 연구소
jungyeul.park@linguist.jussieu.fr

Fast Automatic Bracketing using 2 Level Methods for Korean

Jungyeul Park
Universté Paris VII, UFR Linguistique - Laboratoire de Linguistique Formelle

요 약

이 논문에서는 2 단계 방법을 사용한 한국어 TAG 구문분석기를 구현한다. 2 단계 방법이란 우선 샬로우 파서를 통해 입력 문장을 평면적 구구조로 나눈 다음, 이들 구구조를 대상으로 중심어-부가어를 적용하는 TAG 구문분석 방법을 적용한다. 이런 방법을 통해 TAG 파싱의 복잡도 $O(n^6)$는줄이지 못하지만 입력 문장의 길이를 줄여 빠른 시간내에 파싱을 수행할 수 있다.

1. 들어가는 글

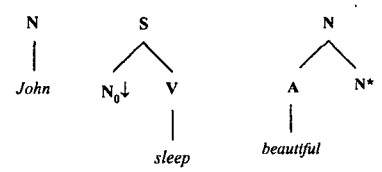
이 논문에서는 2 단계 방법을 사용한 한국어 TAG 파서를 구현한다. 2 단계 방법이란 우선 샬로우 파서를 통해 입력 문장을 평면적 구구조로 나누고, 이들 평면적 구구조를 대상으로 중심어-부가어를 적용하는 TAG 파서를 통해 입력 문장을 심층 분석한다. 샬로우 파서로 심층 분석을 하기 전에 구구조로 나누기 때문에 문장 전체를 동시에 파싱하는 방법보다 입력 문장의 길이를 큰폭으로 줄여주기 때문에 파싱에 있어서 동일한 TAG 파싱 복잡도 $O(n^6)$를 가지지만 파싱 시간을 많이 줄일 수 있다.

이후 이 논문에서는 먼저 TAG 문법의 간단한 개요를 설명한 다음, 샬로우 파싱 및 심층 분석에 대해 설명한다.

2. TAG 문법 개요

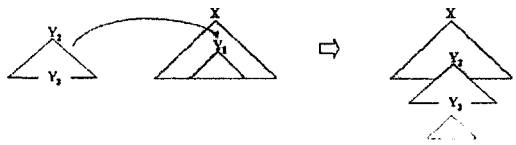
TAG 문법은 트리를 사용해 다시 쓰기 규칙을 구현한 시스템이다[1][2][3]. TAG 형식론은 기본 트리(elementary tree)로 구성되며 기본 트리는 다시 초기 트리(initial tree)와 보조 트리(auxiliary tree)로 구분된다.

다음은 초기 트리와 보조 트리의 간단한 예이다. 아래 그림 1 의 초기 트리 α에서 sleep 의 내부 노드인 V 는 비종단 기호이고 경계 노드는 John 과 sleep 과 같이 종단 기호이거나 N0↓와 같이 트리 외부 치환 (substitution)이 표시된 비종단 기호이다. 보조 트리 β에는 트리 내부 치환 (adjunction)이 가능한 경계 노드 N*은 루트 노드 N 과 동일한 타입이다.

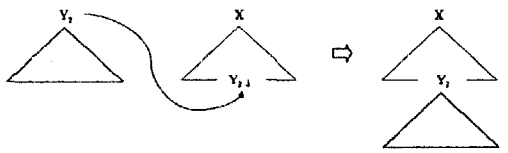


(a) 초기 트리 α (b) 보조 트리 β
그림 1. 초기 트리 및 보조 트리 예

일반적으로 TAG 에서는 트리 내부 치환 및 트리 외부 치환 두 가지 연산을 사용한다. 트리 외부 치환은 초기 트리의 루트 노드가 트리 외부 치환이 표시되어 있는 다른 초기 트리의 비종단 경계 노드로 치환되어 새로운 트리를 생성하는 연산이다. 루트 노드와 비종단 경계 노드는 동일한 타입이어야 한다. 트리 내부 치환은 보조 트리가 초기 트리의 해당 비종단 노드로 치환된다. 위의 그림에서 처럼 보조 트리의 루트 노드와 경계 노드는 동일한 타입이어야 한다. 그림은 트리 외부 치환과 트리 내부 치환을 설명한다.



(a) 트리 내부 치환 연산



(b) 트리 외부 치환 연산
그림 2. TAG 문법에서 사용하는 연산

3. 샬로우 파싱 단계

샬로우 파서는 형태소 분석이 완료된 문장을 입력으로 받아 "평면적 구구조" 분석을 수행한다[4]. 평면적 구구조는 NP 나 VP 와 같은 구문 표지만 평면적으로 표시되고 지배 결속 정보는 포함하지 않는다. 샬로우 파서의 구현 목적은 평면적 구구조 분석으로 파싱 복잡도를 줄이는 것으로 TAG 도출 트리 구현을 위한 입력으로 사용한다.

이 논문에서 제안하는 샬로우 파서를 구현하는 방법은 품사 태깅이 끝난 문장을 조사/어미를 기준으로 분해하여 분석하는 과정으로 이들 형식 형태소를 사용하여 구구조의 통사 범주에 대한 최대 유사도를 계산한다.구구조로 분해한 다음에는 각각의 구구조에서 중심어를 찾고 통사 범주 태그를 부여해야 한다. 다음은 입력 문장 (1)에 대한 샬로우 파싱 결과이다.

(1) 일본 외부성은 즉각 해명 성명을 발표했다.
(S (NP (NNP 일본) (NNG 외무성) (JX 은))
(ADVP (ADV 즉각))

(NP-OBJ (NNG 해명) (NNG 성명) (JKO 을))
(VP (VV 발표했다))
./SFN)

그림 3. 샬로우 파싱 결과

총 270,629 문장에 대해 테스트하여 평균 12.87 어절을 형태소 분석한 결과는 평균 28.60 단어로 증가하지만 샬로우 파서가 만들어내는 평면적 구구조는 평균 구구조의 수는 9.45 개로 줄었다. 이는 동일한 문장에서 어절에 비해 30% 가량 줄어든 결과이며, 일반적으로 심층분석시 사용되는 형태소 수(어휘화 문법)와 비교한다면 70% 가량 줄어든 수치이다.

4. 심층 분석 단계

위에서처럼 샬로우 파싱의 결과는 평면적 구구조를 가지고 있다. TAG 문법은 해당 범주 구에서 중심어 뿐만 아니라 문장의 논항 및 부가항이 구분되어 있기 때문에 평면 구조인 샬로우 파싱 결과를 TAG 문법을 적용하여 논항과 부가항으로 나누어 심층 분석한다.

이 논문에서 사용하는 심층 분석 파서는 [5]에서 제안한 연역 파싱 시스템(deduction parsing system)을 기반으로 [6]의 TAG 파싱 알고리즘을 한국어 시스템에 적용한다. 연역 파싱 시스템이란 파싱을 연역 과정으로 보고 추론 규칙(inference rule)을 사용하여 주어진 문장을 읽어 결과를 도출한다. 일반적으로 추론 규칙은 다음과 같다.

$$\frac{A_1 \ \ldots \ A_k}{B} \quad (A_1 \ldots A_k \text{ 및 B 에 대한 부가 조건})$$

$A_1 \ldots A_k$ 은 이미 차트에 있는 내용이고 부가 조건을 만족할 경우에 아이템 B 를 차트에 추가한다.

연역 파싱 시스템에서 사용하는 TAG 파싱 알고리즘은 최악의 경우에는 $O(|A|\,|A \cup I|\,N\,n^6)$ 복잡도를 갖는다. n 은 입력 문자열의 길이, $|A|$는 보조 트리의 갯수, $|A \cup I|$는 문법에서 사용되는 기본 트리의 갯수(보조 트리와 초기 트리의 합), N 은 기본 트리에서 최대 노드의 수를 의미한다.

TAG 파싱 알고리즘은 아이템을 생성하여 차트에 저장하는 과정으로 기술된다. 아이템 s 는 (α, dot, pos, i, j, k, l, sat?) 같이 정의되며 각 항은 다음과 같다.

- α 는 기본 트리로 초기 트리와 보조 트리로 구성된다.
- *dot* 는 α 에서 주소이다. 다시 말해 트리 α 에서 점의 주소이다.
- *pos* 는 노드에서의 점의 위치로 왼쪽 위, 왼쪽 아래, 오른쪽 아래, 오른쪽 위이다 : $pos \in \{la, lb, rb, ra\}$
- i, j, k, l 는 입력 문자열의 위치를 나타내는 인덱스로 $\{0, \ldots, n\} \cup \{_\}$의 범위를 가진다. n 은 입력 문자열의 길이이며 _는 해당 인덱스가 없는 것을 의미한다. i 와 j 는 항상 인덱스를 가져야 하며, k 와 l 은 동시에 인덱스가 없을 수 있다.
- *sat*?는 트리 α 의 주소 *dot* 의 노드에서 치환 연산이 있는가 없는가에 대한 boolean 값이다. TAG 파싱 알고리즘에서는 $pos = rb$ 일 경우에 *sat*?을 *t* 로 설정한다.

TAG 파싱에서는 ADJUNCTION 연산에서 가장 복잡한 계산을 하게된다. 다음은 ADJUNCTION 연산의 예이다.

$$\frac{[\beta,0,ra,i,j,k,l,nil] \quad [\alpha,dot,rb,j,m,n,k,nil]}{[\alpha,dot,rb,i,m,n,l,true]} \quad \beta \in Adj(\alpha, dot)$$

TAG 문법은 트리 내부 치환 뿐만 아니라 외부 치환 연산도 기본 연산으로 사용한다. 트리에서 외부 치환 연산은 기본 트리의 경계 노드에서 발생한다. 해당 노드에 외부 치환 연산 표시가 있으면 이 노드에서는 내부 치환 연산을 수행할 수 없다. 또한 동일한 비종단 기호를 루트 노드로 가지는 초기 트리만 해당 치환 노드에서 외부 치환될 수 있다.

이 논문에서 사용하는 TAG 파싱 알고리즘은 상향식 파서로 하향식 정보를 사용하기 때문에 SUBSTITION 연산을 PREDICT SUBSTITION 및 COMPLETE SUBSTITION 두가지로 나누어서 고려한다. 트리 α 의 주소 *add* 의 노드가 트리 외부 치환 표시가 있다고 가정하면, 트리 α 의 주소 *add* 의 노드에서 트리 외부 치환 연산이 가능한 초기 트리의 집합을 *Substit*(α, *add*)를 정의할 수 있다. 트리 외부 치환 연산에 대한 제한 사항이 없는 TAG 문법에서는 *Substit*(α, *add*)은 루트 노드가 α(*add*)로 표시된 기본 초기 트리의 집합이다. PREDICT SUBSTITION 연산은 외부 치환 연산이 표시된 노드에서 치환할 수 있는 모든 가능한 초기 트리를 예측한다.

$$\frac{[\alpha,dot,la,i,j,k,l,sat?]}{[\alpha',0,ra,l,\_,\_,l,nil]} \quad \alpha' \in Substit(\alpha, dot)$$

COMPLETE SUBSTITION 연산은 상향식 연산으로 두 개의 아이템을 치환 연산을 통해 결합한다.

$$\frac{[\alpha,nil,ra,l,\_,\_,m,nil] \quad [\alpha',dot',la,i,j,k,l,sat?]}{[\alpha',dot',ra,i,j,k,m,sat?]} \quad \alpha' \in Substit(\alpha, dot)$$

샬로우 파싱의 결과를 심층 분석의 입력으로 받아들이기 때문에 실질적으로 심층 분석 단계에서는 VP 나 ADJP 와 같은 술어 부분만 어휘화 트리를 사용하고 나머지 논항과 부가항은 *np* 나 *advp* 와 같이 샬로우 파싱 단계에서 사용한 통사 범주 태그를 종단 기호로 사용하여 파싱을 수행한다. 따라서 이 논문에서 구현하는 TAG 구문 분석은 엄밀한 의미에서 부분적 LTAG (Partially Lexicalized TAG) 분석이다. 부분적 LTAG 의 가장 큰 문제점으로 술어-논항 관계에서 논항의 의미 정보를 고려치 않기 때문에, 특히 조사가 생략되는 경우에 정확한 논항 분석이 힘들다.

아래 그림은 위의 연산들을 구현하여 얻은 심층 분석 결과이다. 굵게 표시된 노드는 샬로우 파싱 단계에 새로 추가된 노드를 의미한다. 일부 노드는 각 논항 역할을 상술하기 위해 임의로 변경하였다.

```
(S (NP-SBJ (NNP 일본)
           (NNG 외무성)
           (JX 은))
   (VP (AVDP (ADV 즉각))
       (VP (NP-OBJ (NNG 해명)
                   (NNG 성명)
                   (JKO 을))
           (VV 발표했다))))
./SFN)
```

그림 4. 심층 분석 결과

5. 결론

이 논문에서는 2 단계 방법을 사용하여 한국어 TAG 도출트리를 구현하였다. 샬로우 파싱 단계에서 평면적 구구조로 분석하여 심층 분석 단계의 입력 길이를 대폭 줄여 TAG 파싱에서 사용되는 n^6 의 구문 분석 시간을 대폭 줄일 수 있었다. 매우 빠른 시간내에 결과를 얻을 수 있는 샬로우 파싱만으로도 많은 자연어처리 관련 작업을 수행할 수 있으며, 심층분석 결과로는 이후의 의미 분석 뿐만 아니라 대용량 트리뱅크 코퍼스를 자동으로 구축하여 문법 추출이 가능하다.

참고문헌

[1] Joshi, Aravind, L. Levy and M. takahashi. 1975. Tree Adjoining Grammar. In *Journal of Computer and System Science.*

[2] Joshi, Aravind and Yves Schabes. 1994, Tree Adjoining Grammar. In A. Salomma and G. Rogenberg, editors, *Handbook of Formal Languages and Automata.* Springer-Verlag, Herdelberg.

[3] The XTAG Research Group 1999, *A Lexicalized Tree Adjoining Grammar for English*, Technical Report, University of Pennsylvania. http://www.cis.upenn.edu/~xtag.

[4] 박정열 2004, 한국어 샬로우 파서 구현, 파리 7 대학 기술 문서 TR-UFRL-04-009.

[5] Shieber, Stuart M., Yves Schabes, and Fernando Pereira 1995, Principles and Implementation of Deductive Parsing, in *The Journal of Logic Programming.* New York, NY.

[6] Schabes, Yves 1994, Left to Right Parsing of Lexicalized TAG, in *Computational Intelligence*, Volume 10, Number 4, 1994.

부록[1]

Let $G = (\Sigma, NT, I, A, S)$ be a TAG.
Let $a_1 \ldots a_n$ be the input string.
program recognizer
begin
$C = \{ [\alpha, 0, la, 0, \_, \_, nil] \; \alpha \in I, \alpha(0) = S \}$
Apply the following operations on each item in the chart C until no more items can be added to the chart C :

(1) $\dfrac{[\alpha, dot, la, i, j, k, l, nil]}{[\alpha, dot, ra, i, j, k, l+1, nil]} \; \alpha(dot) \in \Sigma, \alpha(dot) = a_{l+1}$

(2) $\dfrac{[\alpha, dot, la, i, j, k, l, nil]}{[\alpha, dot, ra, i, j, k, l, nil]} \; \alpha(dot) \in \Sigma, \alpha(dot) = \varepsilon$

(3) $\dfrac{[\alpha, dot, la, i, j, k, l, nil]}{[\beta, 0, la, l, \_, \_, l, nil]} \; \alpha(dot) \in NT, \beta \in Adj(\alpha, dot)$

(4) $\dfrac{[\alpha, dot, la, i, j, k, l, nil]}{[\alpha, dot, lb, i, j, k, l, nil]} \; \alpha(dot) \in NT, OA(\alpha, dot) = false$

(5) $\dfrac{[\alpha, dot, lb, l, \_, \_, l, nil]}{[\delta, dot', lb, l, \_, \_, l, nil]} \; dot = Foot(\alpha), \alpha \in Adj(\delta, dot')$

(6) $\dfrac{[\alpha, dot, rb, i, j, k, l, nil] \quad [\beta, dot, la, i, \_, \_, i, nil]}{[\beta, dot', rb, i, i, l, l, nil]} \; dot' = Foot(\beta), \beta \in Adj(\alpha, dot)$

(7) $\dfrac{[\alpha, dot, rb, i, j, k, l, true] \quad [\alpha, dot, la, h, j', k', i, sat?]}{[\alpha, dot, ra, h, j \cup j', k \cup k', l, sat?]} \; \alpha(dot) \in NT$

(8) $\dfrac{[\beta, 0, ra, i, j, k, l, nil] \quad [\alpha, dot, rb, j, p, q, k, nil]}{[\alpha, dot, rb, i, p, q, l, true]} \; \beta \in Adj(\alpha, dot)$

If there is an item of the form $[\alpha, 0, la, 0, \_, \_, nil]$ in C with $\alpha(dot) \in I$ and $\alpha(0) = S$ then return acceptance, otherwise return rejection.
end.

[1] [6]